ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA BIOMEDICÍNSKÉHO INŽENÝRSTVÍ

Katedra biomedicínské informatiky

Vytvoření nástroje pro exekuci bioinformatických pipeline a jejich kontrolu

Creation of a tool for bioinformatics pipeline execution and their management

Diplomová práce

Studijní program: Biomedicínská a klinická informatika
Specializace: Softwarové technologie
Vedoucí práce: Ing. Bohuslav Dvorský

Bc. Jaroslav Iha

Kladno 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Iha**	Jméno:	**Jaroslav**	Osobní číslo:	**474300**
Fakulta:	**Fakulta biomedicínského inženýrství**				
Garantující katedra:	**Katedra biomedicínské informatiky**				
Studijní program:	**Biomedicínská a klinická informatika**				

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Vytvoření nástroje pro exekuci bioinformatických pipeline a jejich kontrolu

Název diplomové práce anglicky:

Creation of a tool for bioinformatics pipeline execution and their management

Pokyny pro vypracování:

Diplomová práce adresuje téma automatizace v oblasti bioinformatických procesů pro zpracování dat. Konrétně se jedná o procesy pro zpracování sekvenačních dat nové generace (Next-Generation Sequencing [NGS]). Bioinformatická pipeline je sekvence kroků, které mají vstupy, výstupy a mohou si je předávat mezi sebou. Jednoduchým způsobem, jak vytvořit bioinformatickou pipeline je např. jednotlivé kroky naskriptovat pomocí jazyka Bash. Srozumitelnost, rozšiřitelnost či dokumentace takových pipeline však bývá problematická z povahy vyšší technické náročnosti tohoto řešení. Stejně tak kontrola zdrojů a průběhu pipeline je nelehkou záležitostí vyžadující značné znalosti linuxových systémů. Cílem této práce je vytvořit univerzální nástroj, který umožní spouštět bioinformatické procesy (pipeline), bude poskytovat standardizovanou notaci pro jejich definici, standardizované rozhraní jejich kontroly (spuštění, zastavování apod.) a umožní monitoring jednotlivých instancí pipeline. Dalším cílem je zmapovat možnosti kontroly zdrojů, zejména počtu procesorových jader a operační paměti, pro tento nástroj. Díky předešlému studiu této problematiky víme, že část požadavků splňuje software NextFlow (Di Tommaso, P. et al., 2017), který jsme vybrali jako základ nástroje, který má vzniknout v této práci. Konkrétními podcíli této práce jsou: Popsat možnosti nástroje Nextflow pro exekuci bioinformatických pipeline včetně nasazení. Implementovat či integrovat existující REST API pro ovládání Nextflow a nástroje, jako celku. Implementovat či integrovat existující řešení pro monitoring pipeline. Vytvořit dokumentaci k nástroji a instalační příručku pro uvedení nástroje do provozu.

Seznam doporučené literatury:

[1] Di Tommaso, P., Chatzou, M., Floden, E. W., Barja, P. P., Palumbo, E., & Notredame, C., Nextflow enables reproducible computational workflows, Nature Biotechnology pages, ročník 35, číslo 4, 217
[2] ANTAO Tiago, Bioinformatics with Python Cookbook, ed. Second, 2018, Packt Publishing, 9781789344691
[3] PŘISTOUPILOVÁ Anna, Využití nových metod analýzy genomu ve studiu molekulární podstaty vzácných geneticky podmíněných onemocnění., 2020

Jméno a příjmení vedoucí(ho) diplomové práce:

Ing. Bohuslav Dvorský

Jméno a příjmení konzultanta(ky) diplomové práce:

Mgr. Radim Krupička, Ph.D.

Datum zadání diplomové práce: **14.02.2022**
Platnost zadání diplomové práce: **18.09.2023**

doc. Ing. Zoltán Szabó Ph.D.
vedoucí katedry

prof. MUDr. Jozef Rosina, Ph.D., MBA
děkan

PROHLÁŠENÍ

Prohlašuji, že jsem diplomovou práci s názvem „Vytvoření nástroje pro exekuci bioinformatických pipeline a jejich kontrolu“ vypracoval samostatně a použil k tomu úplný výčet citací použitých pramenů, které uvádím v seznamu přiloženém k diplomové práci.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů.

V Kladně dne 12. 5. 2022 …..……..……...

Bc. Jaroslav Iha

PODĚKOVÁNÍ

Rád bych poděkoval Ing. Bohuslavu Dvorskému za cenné rady, předmětné připomínky a příkladnou spolupráci během vedení mé diplomové práce. Poděkování rovněž patří mé rodině a přítelkyni za jejich nepřetržitou podporu.

ABSTRAKT

Vytvoření nástroje pro exekuci bioinformatických pipeline a jejich kontrolu:

Cílem diplomové práce bylo vytvořit univerzální nástroj pro spouštění, správu a monitorování standardizovaných bioinformatických pipeline. Nástroj byl vytvořen na základě moderních architektonických pravidel a technologií v oblasti softwarového vývoje. Výsledkem práce je funkční řešení, které je snadno integrovatelné do existujících i nově vyvíjených systémů. V závěru jsou formulovány doporučení pro budoucí vývoj a je poukázáno na některé nedostatky a omezení stávajícího řešení.

Klíčová slova

Bioinformatická pipeline, REST API, Nextflow

ABSTRACT

Creation of a tool for bioinformatics pipeline execution and their management:

The aim of the master's thesis was to create a universal tool for running, managing and monitoring standardized bioinformatics pipelines. The tool was created based on modern architectural rules and technologies in the field of software development. The result is a functional solution that can be easily integrated into existing and newly developed systems. Ultimately, recommendations for future development are composed, with the addition of drawing attention to the shortcomings and limitations of the current solution.

Keywords

Bioinformatics pipeline, REST API, Nextflow

Obsah

Seznam symbolů a zkratek

Seznam zkratek

Zkratka	Význam
API	*Application programming interface*
AWS	Webové služby Amazon (*Amazon Web Services*)
CRUD	*Create, Read, Update, Delete*
HTTP	*Hypertext transfer protocol*
JSON	*JavaScript object notation*
REST	*Representational state transfer*
URI	*Uniform resource identifier*
XML	*Extensible markup language*
YAML	*YAML ain't markup language*

1 Úvod

V posledních letech lze sledovat ohromný pokrok v oblasti metod molekulární biologie a je jen otázkou času, kdy se stane sekvenování veškeré genetické informace člověka běžným postupem při diagnostických vyšetřeních. Každým dnem je generováno čím dál tím větší množství genetických dat, což s sebou přináší spoustu výzev. Mezi ně se například řadí ukládání, dostupnost a šíření těchto dat, zvyšující se nároky na výpočetní kapacitu, reprodukovatelnost výsledků, rostoucí komplexita řešení atd. Bioinformatika se jako vědní obor těmito problémy zabývá, a tím roste i její význam na prahu mezi biologií a informatikou.

Zpracování a analýza sekvenačních dat není jednoduchým úkolem, pro který existuje velké množství nástrojů. Bohužel většina nástrojů je vyvíjena a používána v úzkém kruhu uživatelů, často i pouze na jednom stroji, a tím pádem mají složitou, často nepřenositelnou konfiguraci, která závisí na nastavení hostujícího prostředí. Příkladem je akademická půda, kde dochází k vývoji experimentální cestou a výsledný produkt je velmi složité instalovat, konfigurovat nebo nasazovat. Řešení také obsahuje velké množství závislostí (knihovny, systémové nástroje, kompilátory atd.). Problémem jsou i heterogenní exekuční platformy, kdy rozhodně nelze uvažovat o použití stejného řešení jak na superpočítači, tak na laptopu. Další překážkou je nízká úroveň automatizace každodenních úkonů. Jednotlivé analýzy si mezi sebou dokážou předat dílčí výstupy a vstupy jen s pomocí uživatele. Navíc je integrování kteréhokoliv dnešního nástroje, do již stávajícího systému velice komplikovanou záležitostí a doprovází ho celá řada problémů (plánování procesů, paralelní běhy, monitorování instancí, ovládání atd.)

Kvůli problematické dokumentaci a výše zmíněným aspektům vzniká otázka, jak moc náročná je reprodukovatelnost výsledků. V [1] se autoři tímto tématem zabývali a jejich časový odhad pro reprodukování výsledků bioinformatické studie člověkem se základními znalostmi z bioinformatiky je přibližně 280 hodin.

Tato práce se zabývá vytvořením řešení, které je postavené na nástroji Nextflow [2] a poskytuje komplexní rozhraní nad správou standardizovaných bioinformatických pipeline. Navíc je ho možné snadno zaintegrovat do již stávajících i nově vznikajících bioinformatických aplikací.

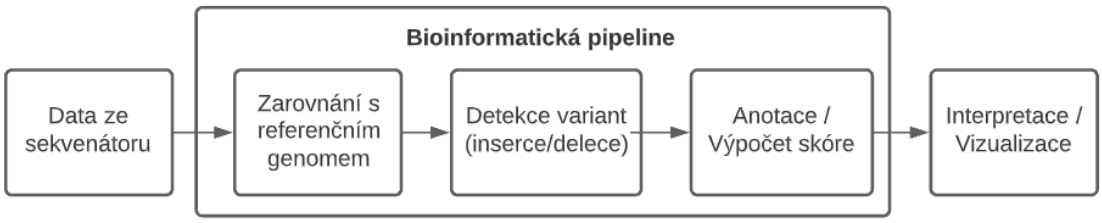
2 Přehled současného stavu

Tato kapitola obsahuje vysvětlení základních pojmů, popis a analýzu moderních technologií, nástrojů a architektur, které jsou nejen aktuálními trendy v oblasti informačních technologií, ale jsou i nedílnou součástí této práce.

2.1 Bioinformatická pipeline

Pojem pipeline obecně označuje předepsanou sadu kroků potřebnou k převedení nezpracovaných surových dat do interpretovatelné podoby. Bioinformatická pipeline popisuje sekvenci na sebe navazujících procesů, kterou lze použít k analyzování sekvenčních dat. Většinou se v každé části data zpracovávají pomocí jedné metody/algoritmu. Jednotlivé kroky si mezi sebou mohou předávat vstupy i výstupy. Lze si to představit jako několik zřetězených „jednoduchých" programů, kdy každý nad daty provede svoji definovanou operaci a výstup předá dalšímu. [3]

Na obrázku 2.1 jsou zobrazeny jednotlivé kroky, které mohou být součástí jednoduché bioinformatické pipeline.



Obrázek 2.1: Příklad bioinformatické pipeline. Zdroj: vlastní

2.2 REST API

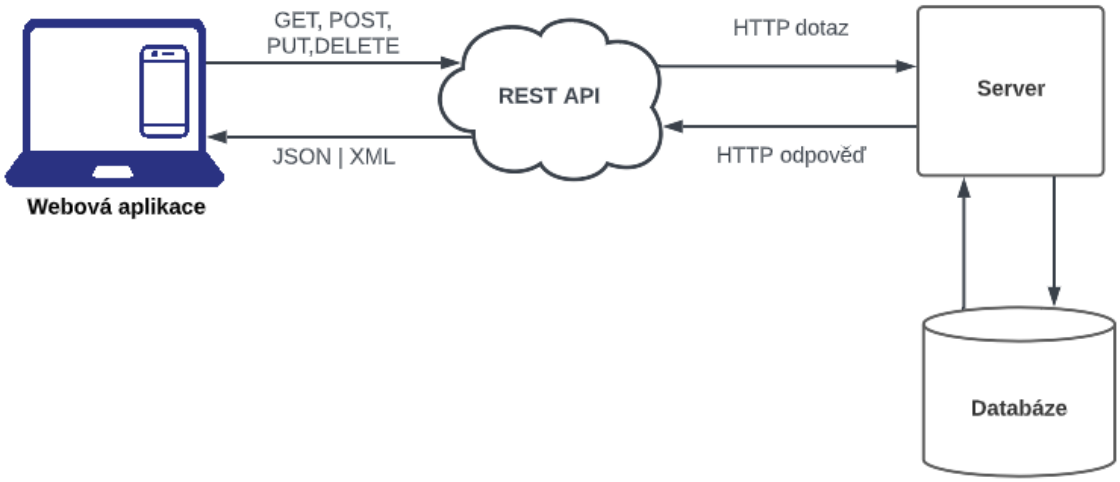
REST API (také známé pod názvem RESTful API) je označení pro aplikační rozhraní podléhající zásadám architektury REST. Jde o snadný způsob integrace aplikací, který se osvědčil jako nejběžnější metoda pro spojování komponent v rámci mikroslužbové architektury. REST je zkratkou pro REpresentational State Transfer a tvůrcem je počítačový vědec Dr. Roy Fielding. [4]

API je souhrn pravidel a definic udávající, jak se mohou zařízení nebo aplikace vzájemně propojit a komunikovat mezi sebou. Jedná se o jakousi smlouvu mezi poskytovatelem informací a uživatelem ustanovující obsah požadovaný od konzumenta (požadavek) a producenta (odpověď). Pokud chce uživatel komunikovat se systémem za účelem získání informací, API rozhraní mu pomůže sdělit požadavky na systém, aby jim mohl v pořádku porozumět a splnit je. Organizace hojně využívají API pro bezpečné sdílení informací a zdrojů. Je možné kontrolovat k jakým informacím a metodám má kdo přístup. [5] Jako příklad si lze představit meteorologickou aplikaci, jejíž API návrh by

definoval, že uživatel poskytne název svého města a producentova odpověď vrátí průměrnou teplotu pro danou lokalitu.

REST je sada architektonických pravidel pro návrh API, která mohou být implementována různými způsoby a jsou navržena tak, aby využívala předností již existujících protokolů.

Obrázek 2.2 znázorňuje zasazení REST API do kontextu webové služby.

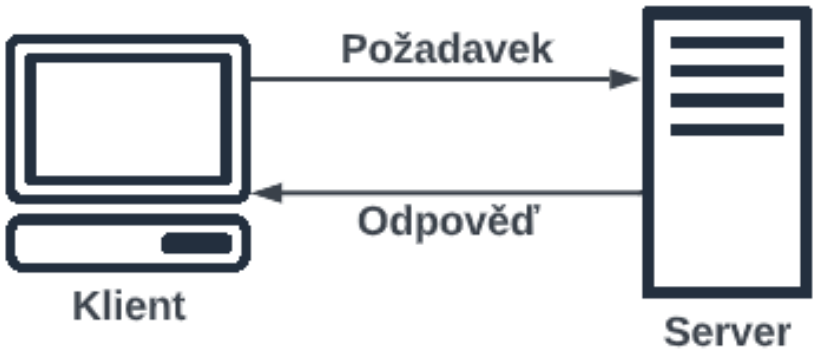


Obrázek 2.2: REST API. Převzato a upraveno podle zdroje: [6]

Aby mohlo být rozhraní považováno za REST, musí splňovat následující podmínky:

Klient-server architektura

Architektura klient-server je jednou z nejpoužívanějších architektur v rámci webových aplikací. Server jako komponenta poskytuje službu a naslouchá požadavkům. Klient se k serveru připojuje a pomocí požadavků využívá serverové služby. Každý požadavek je serverem zpracován a klientovi je odeslána příslušná odpověď. Na obrázku 2.3 je znázorněné zjednodušené schéma klient-server architektury. [4]



Obrázek 2.3: Klient-server architektura. Převzato a upraveno podle zdroje: [4]

Zdroje a jejich reprezentace

Architektura REST je datově orientována a základními prvky jsou zdroje. Zdroj může představovat jakoukoliv informaci, kterou lze pojmenovat a která může být reprezentována v datové podobě. V aplikaci online knihkupectví je zdrojem například kniha, objednávka nebo registrovaný zákazník. [4]

Každý zdroj musí být jednoznačně adresovatelný minimálně jedním identifikátorem URI (*Uniform Resource Identifier*). Pomocí tohoto identifikátoru je klient schopen se zdrojem manipulovat. Pro přístup k určitému zdroji slouží vždy jeho reprezentace. I když jde pro reprezentaci zdrojů a manipulaci s nimi využít množství formátů (JSON, HTML, XML, Python, prostý text atd.), dvěma nejvyužívanějšími jsou JSON a XML, protože jsou snadno čitelné jak pro počítač, tak pro člověka. [6]

Na obrázcích 2.4 a 2.5 je příklad reprezentace knihy jako zdroje ve dvou různých formátech (JSON a XML), která by mohla být získána zasláním požadavku na URI `http://www.isbn.com/books/9780547260693`.

```
{
    "book":{
        "title":"The Little Prince",
        "isbn":"9780547260693",
        "author":"Antoine de Saint-Exupéry",
        "pages":"64",
        "published":"12/October/2009"
    }
}
```

Obrázek 2.4: Reprezentace zdroje ve formátu JSON. Zdroj: vlastní

```
<book>
  <title>The Little Prince</title>
  <isbn>9780547260693</isbn>
  <author>Antoine de Saint-Exupéry</author>
  <pages>64</pages>
  <published>12/October/2009</published>
</book>
```

Obrázek 2.5: Reprezentace zdroje ve formátu XML. Zdroj: vlastní

Uniformní rozhraní

Stěžejním prvkem k oddělení klienta od serveru je uniformní rozhraní. Jestliže je rozhraní u komponent všeobecné, umožňuje to oddělit jejich implementace od služeb které poskytují. Vývoj jednotlivých komponent poté může být plně nezávislý na změnách v jiných částech systému.

Všechny požadavky na stejný zdroj by měly vypadat stejně, bez ohledu na to, odkud požadavek pochází. Uniformní rozhraní by mělo pro komunikaci mezi klientem a serverem poskytovat standardizované prostředky, nezávislé na architektuře. Odpovědi vrácené serverem musejí mít dostatek informací k tomu, aby klient věděl, jak je zpracovat. Po přístupu ke zdroji by se měl pomocí hypertextových odkazů (hypertext – text obsahující odkazy na další texty) klient dozvědět další aktuálně dostupné akce, které může provést, a tudíž pro komunikaci nepotřebuje žádné dodatečné informace. [7]

Bezstavovost

Komunikace skrze REST API musí být bezstavová. Volání lze provádět nezávisle na sobě a každé obsahuje veškerá data potřebná k úspěšnému dokončení. Rozhraní by se při zpracování požadavku mělo spoléhat pouze na data poskytnutá v samotném volání a nespoléhat se na data uložená na serveru. Kvůli snížení požadavků na paměť může být jakýkoliv stav komunikace uložen na straně klienta, ale ne na serveru. [4]

Správa mezipaměti

Tím, že je rozhraní bezstavové, se může znatelně zvýšit počet zpracovávaných požadavků i odchozích odpovědí. Proto by mělo být REST API navrženo tak, aby podporovalo ukládání dat, která uložit lze, do mezipaměti (tzv. *caching*). Odpověď by měla udávat, jestli ji je možné uložit do klientovi mezipaměti či nikoli a pokud ano, tak za jak dlouho vyprší její platnost. To částečně zamezuje situaci, aby se data v mezpaměti nelišila od dat, která by klient získal aktuálním požadavkem na server. Správným využitím mezipaměti se snižuje počet interakcí s rozhraním, a tím se zvyšuje jeho efektivita. [4]

Vrstvený systém

Ve vrstveném systému má každá vrstva specifickou funkci a odpovědnost. Vrstvy jsou samostatně oddělené, ale interagují mezi sebou, což vytváří hierarchii, která zlepšuje efektivitu a bezpečnost systému. Vrstvení navíc umožňuje separování nových komponent od zastaralých, zjednodušení existujících komponent přesunutím jejich méně využívaných funkcí do sdílených prostředníků. Co nejvolnější propojení modulů přispívá k delší životnosti systému a přináší značné bezpečnostní výhody. Případné útoky se dají zastavit v různých vrstvách a zabrání se jim tak v přístupu ke skutečné architektuře serveru. [4]

2.2.1 Protokol HTTP

Ačkoli je možné REST použít v kombinaci s téměř jakýmkoliv protokolem, pro webová API se nejčastěji využívá protokol HTTP. Při komunikaci přes HTTP se nad zdroji provádějí standardní operace jako vytváření, čtení, upravování a mazání, známé pod zkratkou CRUD (create, read/retrieve, update, delete).

Metody

V protokolu HTTP pro tyto operace slouží metody: [8]

- GET – načtení zdroje
- POST – vytvoření nového zdroje
- PUT (PATCH) – aktualizace zdroje / vytvoření, pokud ještě neexistuje
- DELETE – odstranění zdroje

Při upravování zdroje pomocí metody PUT se nahrazuje celý zdroj, kdežto při použití metody PATCH lze aktualizovat zdroj jen částečně, ale nelze ho vytvořit, pokud ještě neexistuje.

Stavové kódy

Protokol HTTP definuje stavové kódy jako odpověď na své metody. Kódy jsou reprezentovány třímístným číslem a popisem. Rozdělují se do několika kategorií: [9]

- 1xx – Informační
- 2xx – Úspěšné
- 3xx – Přesměrování
- 4xx – Chyba na klientově straně
- 5xx – Chyba na straně serveru

Nejčastěji používanými stavovými kódy jsou: [9]

- 200 OK – Požadavek byl úspěšný.
- 201 Created – Úprava či vytvoření zdroje byla úspěšná.
- 204 No Content – Požadavek byl úspěšný, ale není vráceno žádné tělo odpovědi.
- 400 Bad Request – Server nedokázal provést požadavek, kvůli chybě v jeho syntaxi.
- 401 Unauthorized – Server potřebuje k provedení požadavku ověřit identitu klienta.
- 403 Forbidden – Klient nemá dostatečná práva pro provedení požadavku.
- 404 Not Found – Zdroj nenalezen
- 500 Internal Server Error – Neočekávaná chyba vnitřní služby serveru.

Hlavičky

Dalším důležitým prvkem při komunikaci skrz protokol HTTP jsou hlavičky a parametry. Obsahují důležité a někdy vyžadované informace jako autorizační tokeny, metadata požadavku, informace o identifikátorech, cookies a další. Využití hlaviček požadavků a odpovědí musí být samozřejmostí v dobře navržených RESTových rozhraních. [10]

Tabulky 2.1 a 2.2 obsahují příklady často používaných hlaviček.

Hlavička	Popis	Příklad
`Accept`	Akceptovatelný formát odpovědi	`Accept: application/json`
`Content-Type`	Formát těla požadavku	`Content-Type: application/json`
`Connection`	Nastavení pro udržení připojení	`Connection: keep-alive`
`Host`	Jméno hostitelského serveru	`Host: www.example.com`

Tabulka 2.1: Hlavičky požadavků. Zdroj: vlastní

Hlavička	Popis	Příklad
`Date`	Datum a čas odeslání odpovědi	`Date: Wed, 16 Oct 2021 09:28:00 GMT`
`Content-Length`	Velikost těla odpovědi v bytech	`Content-Length: 6553`
`Last-Modified`	Datum a čas poslední změny zdroje	`Last-Modified: Tue, 15 Oct 2021 17:45:26 GMT`
`Content-Type`	Formát těla odpovědi	`Content-Type: application/xml`

Tabulka 2.2: Hlavičky odpovědí. Zdroj: vlastní

2.3 Kontejnerizace

Pojem kontejnerizace je v oblasti softwarového vývoje používán pro zabalení kódu aplikace se všemi jeho potřebnými komponentami jako jsou knihovny, frameworky a další závislosti do jediného izolovaného spustitelného objektu zvaného **kontejner**. Aplikace se poté dají snadněji přenášet a konzistentně spouštět bez závislosti na prostředí a na jakékoliv infrastruktuře. Kontejner představuje jakousi bublinu kolem aplikace udržující ji nezávislou na svém okolí. V podstatě se jedná o plně funkční přenositelné výpočetní prostředí. Vývojáři díky kontejnerizaci dokážou vyvíjet a nasazovat aplikace rychleji a bezpečněji. Tradičními metodami je kód vyvíjen na jedné platformě nebo operačním systému, což znatelně komplikuje jeho pozdější přesun, protože aplikace nemusí být kompatibilní s novým prostředím. Zabalení aplikace do kontejneru odstraňuje tento problém, jelikož si s sebou nese vše potřebné (konfigurační soubory, knihovny, závislosti) k úspěšnému spuštění. Po zabalení jde aplikace jednoduše provozovat napříč různými platformami nebo například v cloudu. [11]

2.3.1 Výhody kontejnerizace

Kontejnerizace oproti klasickým metodám přináší při vývoji softwaru velké množství výhod. Některými z nich jsou:

Přenositelnost

Protože kontejner není závislý na hostitelském operačním systému, tak mezi nimi existuje jakási forma abstrakce, díky níž je kontejner snadno přenosný a je ho možné beze změn spouštět na jakékoliv platformě. [12]

Rychlost

Pro kontejnery se často používá označení „lehké“ (*lightweight*). Pochází to z jejich schopnosti sdílet jádro operačního systému hostitelského počítače. Eliminuje se tak potřeba samostatného operačního systému pro každý kontejner. Protože není potřeba zavádět žádný operační systém, významně se zrychluje doba spouštění, snižují se případné náklady a zvyšuje se efektivita serveru. [12]

Izolování chyb

Kontejnerizované aplikace fungují samostatně, izolovaně a nezávisle na ostatních. V případě poruchy jednoho kontejneru tato situace neovlivní další provoz ostatních kontejnerů. Identifikování a opravování chyb může probíhat v rámci jednoho kontejneru, aniž by došlo k pozastavení ostatních kontejnerů. [12]

Efektivita

Nejenže aplikace běžící v kontejnerových prostředích sdílejí jádro operačního systému počítače, ale i aplikační vrstvy v kontejneru se dají sdílet napříč jinými kontejnery. V základu zabírají kontejnery oproti virtuálním počítačům znatelně menší kapacitu a potřebují kratší dobu pro spuštění, což umožňuje provoz většího množství kontejnerů na identické výpočetní kapacitě, jaká by byla potřebná například pro jeden virtuální stroj. [12]

Zabezpečení

Izolování jednotlivých kontejnerů s aplikacemi už z principu brání vniknutí škodlivého kódu a případnému ovlivnění ostatních kontejnerů nebo hostitelského systému. Navíc je možné vytvářet bezpečnostní opatření, která blokují vstupování nežádoucích komponent dovnitř kontejnerů nebo omezují komunikaci s nepotřebnými zdroji. [12]

2.3.2 Kontejnery versus virtuální stroje

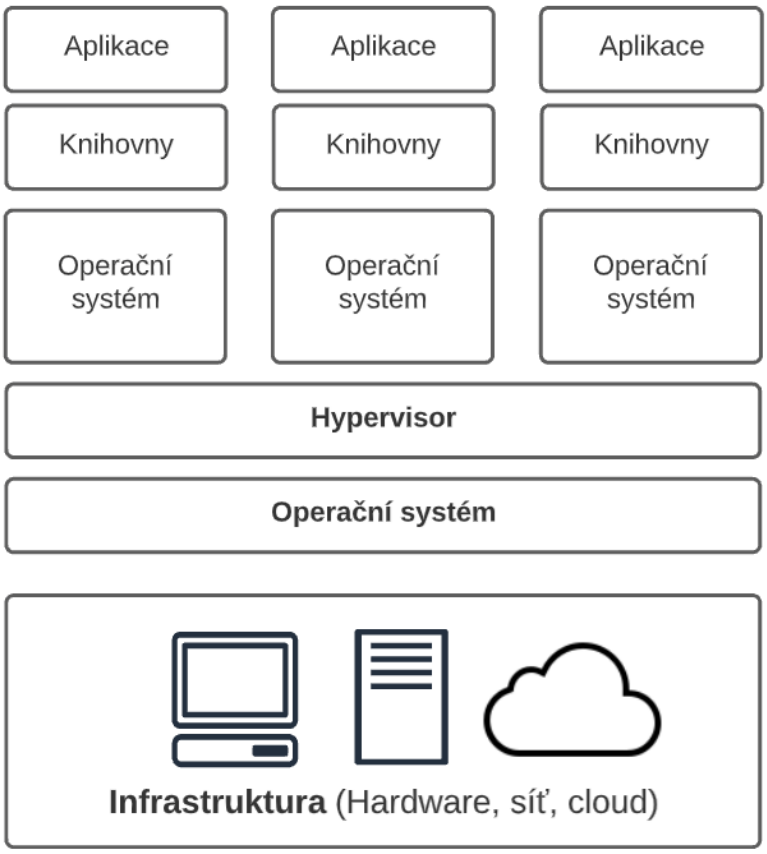
Technologie kontejnerizace a virtualizace bývají často srovnávány, jelikož zvyšují výpočetní efektivitu tím, že umožňují plnou izolaci aplikací, aby mohli fungovat ve vícero

prostředích. Ukazuje se, že v klíčových rozdílech významně převládají výhody kontejnerizace a stává se preferovanější technologií.

Virtualizace

Virtuální stroj je virtuální prostředí fungující jako počítačový systém s vlastními prostředky (procesor, paměť, úložiště, síťové rozhraní), které je vytvořené na fyzickém infrastruktuře. Virtualizace umožňuje paralelní běh většího počtu operačních systémů a aplikací pomocí sdílení zdrojů jednoho fyzického hardwaru. Důležitým prvkem virtualizace je software zvaný hypervisor, ten odděluje fyzické zdroje od virtuálních prostředí, jež tyto zdroje vyžadují. Hypervisor operuje nad operačním systémem tak, že bere fyzické prostředky počítače a přerozděluje je, aby je mohla využívat virtuální prostředí. [13]

Jak vypadá architektura virtuálního stroje je zobrazeno na obrázku 2.6.



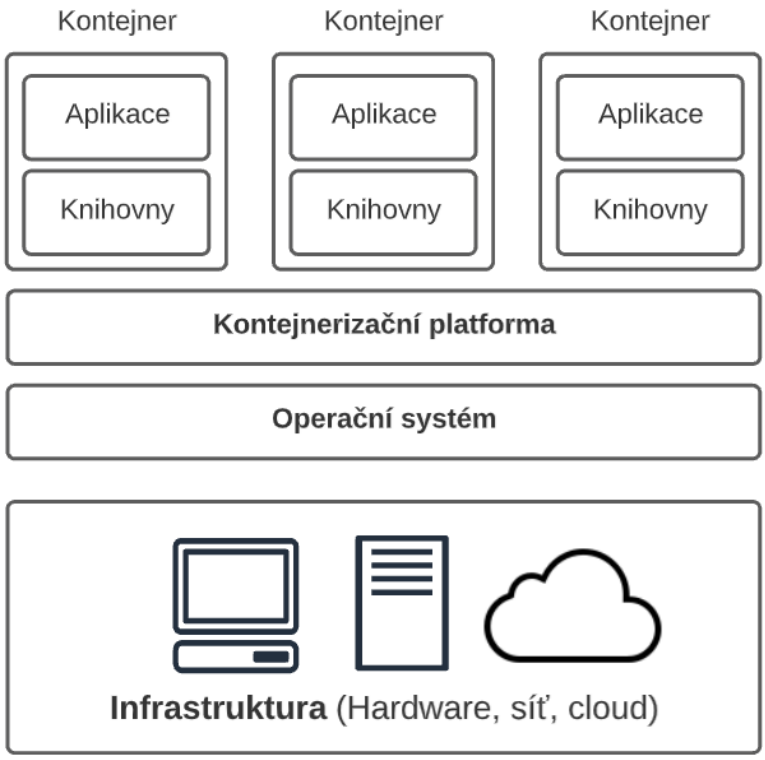
Obrázek 2.6: Architektura virtuálního stroje. Převzato a upraveno podle zdroje: [13]

Kontejnerizace

Kontejnerizace oproti virtualizaci dokáže využívat výpočetní prostředky o něco efektivněji. Kontejner sdružuje aplikační kód spolu se všemi knihovnami a konfiguračními soubory potřebnými k jeho úspěšnému spuštění. Na rozdíl od virtuálních strojů každý kontejner neběží v samostatné kopii operačního systému. Místo toho je platforma pro běh kontejnerů nainstalována přímo na operační systém hostitelského

systému a funguje jako prostředník, přes kterého veškeré kontejnery sdílejí stejný operační systém. [14]

Obrázek 2.7 znázorňuje architekturu kontejnerové technologie.



Obrázek 2.7: Architektura kontejnerizace. Převzato a upraveno podle zdroje: [14]

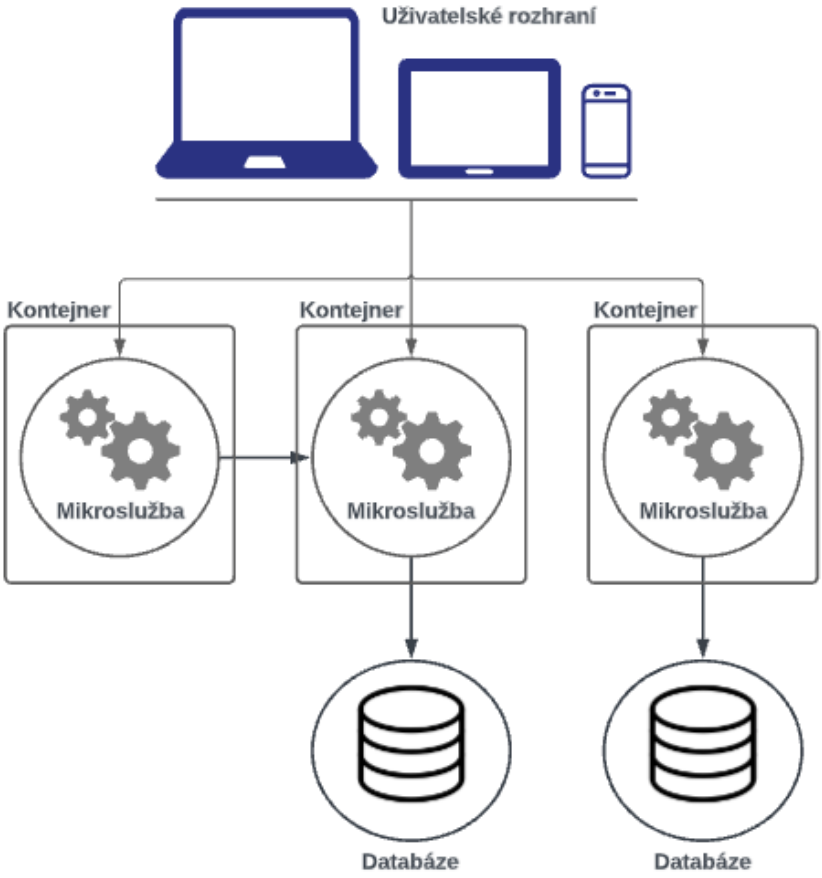
2.3.3 Kontejnery a mikroslužby

Kontejnery se často používají k zabalení jednotlivých funkcionalit, které jsou určeny pro provádění specifických úloh. Takto charakteristické funkce se též nazývají jako mikroslužby. Mikroslužby jsou vynikajícím způsobem, jak přistupovat k vývoji a správě aplikací, oproti dříve využívané monolitické architektuře, která sdružuje celou aplikaci i s uživatelským rozhraním a databází do jediného bloku na jediné serverové platformě. V rámci mikroslužbové architektury je komplexní aplikace rozdělena na řadu menších, více specializovaných částí s vlastní databází a obchodní logikou. Komunikace mezi jednotlivými mikroslužbami poté nejčastěji probíhá přes RESTová rozhraní pomocí protokolu HTTP. Velikou výhodou je, že při vývoji je možné soustředit práci na konkrétní oblast aplikace, aniž by to mělo jakýkoliv vliv na její celkový výkon. Aplikace zůstávají při aktualizacích nebo opravách chyb v provozu, což umožňuje rychlejší vývoj, testování a nasazení. [15] [16]

Kontejnerizace a mikroslužby jsou postupy vývoje softwaru založené na obdobném konceptu, při němž oba rozdělují aplikace na soubory menších komponent, které jsou efektivnější, lépe přenosné a snadněji spravovatelné. Z tohoto důvodu fungují kontejnery

a mikroslužby velice dobře pospolu. Kontejnery poskytují lehké zapouzdření jakékoliv aplikace a jedná-li se o mikroslužbu, pak získává veškeré výhody kontejnerizace (izolace chyb, zabezpečení atd.).

Obrázek 2.8 zobrazuje mikroslužbovou architekturu s použitím technolgie kontejnerů.



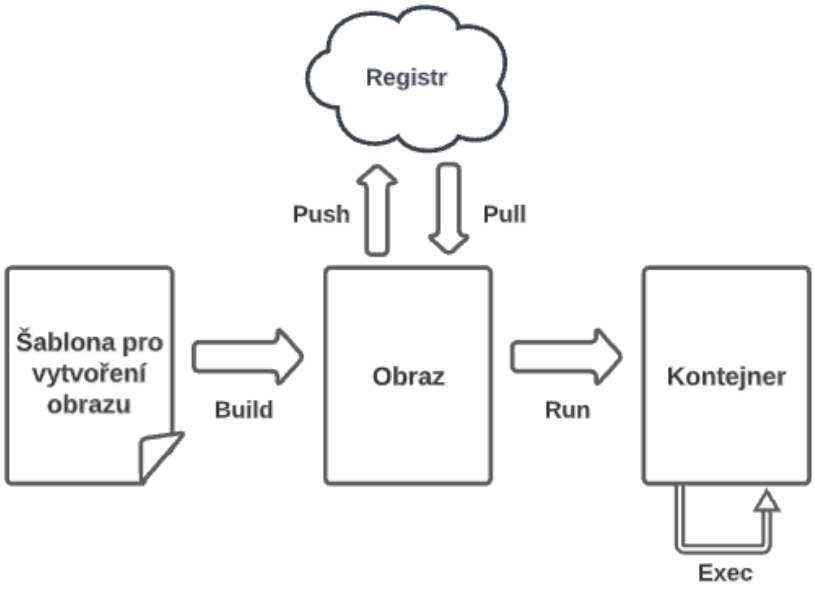
Obrázek 2.8: Architektura mikroslužeb. Převzato a upraveno podle zdroje: [16]

2.3.4 Obecný postup kontejnerizace

Postup pro vytváření a spouštění kontejnerů je téměř identický napříč všemi nástroji pro kontejnerizaci. Základem jsou specifické konfigurační soubory, v tomto případě zvané šablony, které jsou použity pro sestavení obrazu. Obraz obsahuje veškeré knihovny a binární soubory aplikací, které mají být spuštěny jako kontejner. Konfigurační soubory většinou začínají definováním Linuxové distribuce jakožto operačního systému pro kontejner. Poté lze během procesu sestavování načíst, nakonfigurovat a zkompilovat další zdroje. Většina nástrojů poskytuje vlastní registr, službu, kde je možné ukládat nebo získávat již vytvořené obrazy.

Obecný postup kontejnerizace je vyobrazen na obrázku 2.9. Příkazem `build` dojde k vytvoření obrazu ze šablony. Obrazy je možné uchovávat lokálně anebo získávat ze vzdáleného či místního registru. Pomocí příkazu `run` se z obrazu spustí instance

kontejneru. Z jednoho obrazu lze spustit více kontejnerových instancí. Příkaz `exec` umožňuje spouštění příkazů v rámci běžícího kontejneru.



Obrázek 2.9: Obecný postup kontejnerizace. Zdroj: vlastní

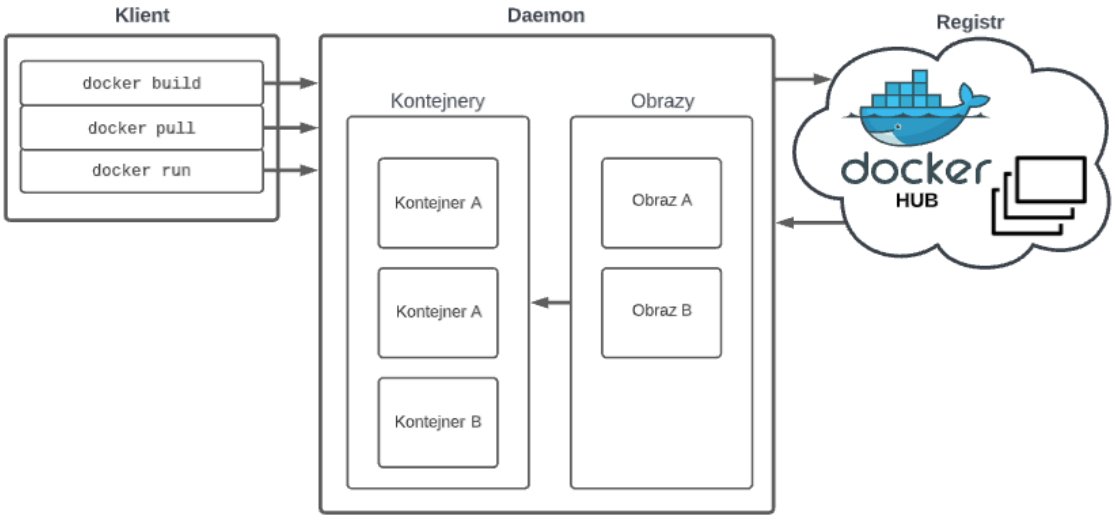
2.3.5 Docker

Docker je otevřená platforma pro vývoj, spouštění a správu kontejnerizovaných aplikací, která se se brzy po vydání (2013) stala standardem v oblasti kontejnerizace. Docker je vytvořen v programovacím jazyce Go a ke své funkčnosti využívá několik funkcí linuxového jádra. Například při vytváření kontejnerů používá technologii jmenných prostorů (*namespaces*). Když je kontejner spuštěn, Docker pro něj vytvoří kolekci jmenných prostorů, které poskytují vrstvu izolace pro běžící procesy uvnitř kontejneru. [17]

Architektura platformy Docker

Docker je postaven na architektuře klient-server. Docker klient komunikuje přes síťové rozhraní pomocí REST API s analogií serveru zvanou Docker daemon, která zajišťuje vytváření, provoz a distribuci obrazů a kontejnerů. Klient i daemon mohou fungovat na stejném systému nebo je možné připojit klienta ke vzdálenému daemonu. Alternativou k Docker klientovi je Docker Compose, ten poskytuje funkcionalitu k práci s aplikacemi složenými z více kontejnerů. [17]

Obrázek 2.10 popisuje obecnou architekturu platformy Docker.



Obrázek 2.10: Architektura platformy Docker. Převzato a upraveno podle zdroje: [17]

- **Docker daemon**

Daemon označuje program běžící nepřetržitě na pozadí systému, který zpracovává obdržené požadavky. Docker daemon naslouchá příchozím požadavkům od klienta a na jejich základě spravuje Docker objekty, jako jsou obrazy, kontejnery a sítě. Za účelem správy služeb může také Docker daemon komunikovat i s jinými daemony. [17]

- **Docker klient**

Primárním způsobem interakce mezi uživatelem a Dockerem je Docker klient. Použité příkazy, jako například `docker run`, odešle klient přes API daemonovi, který danou operaci s objekty provede. Výhodou klienta je, že může komunikovat s více než jedním daemonem najednou. [17]

- **Docker registr**

Docker registr lze přirovnat k databázi Docker obrazů. Docker provozuje vlastní veřejně přístupný registr pro sdílení obrazů zvaný Docker Hub. V základu je Docker nakonfigurovaný tak, aby vyhledával obrazy právě na Docker Hubu. Pro zvýšení bezpečnosti je možné provozovat vlastní soukromý registr. [17]

Použitím příkazů `docker push` nebo `docker pull` se požadované obrazy nahrávají či stahují z uživatelem nakonfigurovaného registru.

Docker objekty

Docker operuje s celou řadou objektů, níže je přehled těch nejčastěji využívaných.

- **Obrazy**

Docker obrazy jsou určené pouze pro čtení a obsahují pokyny pro vytvoření Docker kontejnerů. Často bývají založeny na jiném Docker obrazu, který dále upravují nebo přizpůsobují. Základem obrazu může například být obraz distribuce Linuxu Ubuntu,

následně obraz nainstaluje webový server Apache a webovou aplikaci se všemi konfiguračními detaily, které jsou potřebné pro její úspěšné spuštění. [17]

Uživatelé si mohou vytvářet své vlastní obrazy nebo mohou použít ty vytvořené a publikované v registru jinými uživateli. Pro vytvoření vlastního obrazu slouží soubor zvaný **Dockerfile** (šablona – viz obrázek 2.9). V Dockerfile se pomocí jednoduché syntaxe definují kroky potřebné k vytvoření obrazu a jeho spuštění. Každá instrukce v souboru vytvoří vlastní vrstvu a pokud se Dockerfile změní, tak se při opakovaném sestavení obrazu znovu vytvoří pouze pozměněné vrstvy. Tato skutečnost je jedním z důvodů, proč je kontejnerizace rychlejší a méně náročná na paměťové požadavky ve srovnání s ostatními virtualizačními technologiemi. [17]

- **Kontejnery**

Docker kontejnery jsou spustitelnými instancemi Docker obrazů, které se dají vytvářet, spouštět, zastavovat, přesouvat a mazat pomocí Docker rozhraní. V základním nastavení jsou od sebe kontejnery relativně dobře izolované. To, jak jsou kontejnery navzájem izolované, lze řídit přes Docker sítě. Ke kontejneru je možné připojit úložiště nebo vytvořit nový obraz podle jeho současného stavu. Pokud se kontejner odstraní, ztratí se i všechny provedené změny s výjimkou těch uložených v trvalém úložišti. [17]

2.3.6 Singularity

Singularity je alternativou k Dockeru, jedná se také o open-source kontejnerizační platformu. Byla vyvinuta v Lawrence Berkeley National Laboratory s cílem zvýšit mobilitu výpočetní síly ve vědecké oblasti a umožnuje vytvářet a spouštět kontejnery lehce přenositelným a reprodukovatelným způsobem. Architektura Singularity zajišťuje, aby běžní uživatelé mohli bezpečně operovat s kontejnery na klastrovém systému HPC (*High Peformance Computing*) bez možnosti práv root (nejvyšší oprávnění). Singularity oproti platformě Docker používá jednodušší přístup bez procesu daemona. Veškeré operace jsou prováděny pouze s uživatelskými právy a k izolování procesů je rovněž využita technologie jmenných prostorů (*namespaces*). Výhodou Singularity je, že může používat jak Singularity, tak i Docker obrazy, buď přímo pro spouštění nebo také jako základ v šabloně pro sestavení obrazu. Podobně jako Docker disponuje Singularity veřejným registrem pro obrazy. [18]

Přednosti Singularity: [19]

- Použití kryptografických podpisů, dešifrování v paměti a imutabilního (neměnného) formátu obrazů zajišťuje ověřitelnou reprodukovatelnost a bezpečnost.
- Snadné používání procesoru grafické karty (*GPU*), vysokorychlostních sítí a paralelních souborových systémů již v rámci výchozího nastavení.

- Jednoduchá přenositelnost a sdílení díky formátování kontejneru do jediného souboru.
- Efektivní model zabezpečení. Uživatel má stejná práva uvnitř kontejneru i mimo něj a v základním nastavení není možné získat rozšířené oprávnění na hostitelském systému.

2.4 Orchestrace kontejnerů

Některé rozsáhlejší aplikace postavené na technologii kontejnerů a mikroslužeb se mohou skládat z desítek, stovek až tisíců kontejnerů. Potřeba, aby všechny běžely a byly spravovány naráz, začne vyžadovat obrovské úsilí. Orchestrace kontejnerů je skvělým způsobem, jak automatizovat nasazení, správu, škálování a síťové propojení kontejnerů. Orchestrace kontejnerů pomáhá nasadit stejnou aplikaci v odlišných prostředích, aniž by se musela jakkoli upravovat.

2.4.1 Proces orchestrace

Většina nástrojů pro orchestraci kontejnerů se odlišuje ve svých metodologiích, ale jejich princip zůstává podobný. Základem je konfigurační soubor (nejčastěji ve formátu YAML nebo JSON), který definuje požadovanou podobu konfigurace. Nástroje mají implementovanou vlastní inteligenci, s jejíž pomocí se snaží požadovaného stavu dosáhnout. [20] [21] Konfigurační soubor typicky obsahuje: [22]

- Definici obrazů utvářející aplikaci a kde se nacházejí (lokálně nebo některý z registrů)
- Definici a zabezpečení síťových propojení mezi kontejnery
- Informace o poskytování paměti a dalších zdrojů kontejnerům

Orchestrační nástroj na základě požadavků a omezení uvedených v konfiguračním souboru naplánuje nasazení kontejnerů. Po nasazení nástroj spravuje a řídí životní cyklus aplikace podle šablony definující kontejner (např. Dockerfile) což zahrnuje: [22]

- Škálování, vyvažování zátěže a alokaci zdrojů napříč kontejnery
- Zajištění dostupnosti a výkonu aplikace i v případě výpadku nebo nedostatku systémových prostředků
- Sběr a ukládání informací o akruálním stavu a výkonu aplikace

2.4.2 Výhody orchestrace

Orchestrace kontejnerů je klíčem k práci s kontejnery a umožňuje na maximum využít jejich výhod. Včetně toho nabízí i vlastní pozitiva: [20]

- **Zjednodušení operací**: Toto je nejdůležitější přínos orchestrace kontejnerů. Velké množství kontejnerů se rychle stává složitým na údržbu, která se může

vymknout kontrole. Díky orchestraci se údržba stává zvládnutelnou, protože automatizuje velkou část potřebné práce.

- **Odolnost**: Nástroje pro orchestraci a správu kontejnerů mohou automaticky restartovat nebo škálovat kontejnery, čímž zvyšují odolnost aplikace.
- **Zabezpečení**: Automatizovaný přístup ke složitým úlohám pomáhá udržovat vyšší úroveň bezpečnosti aplikací tím, že snižuje nebo dokonce eliminuje riziko lidského pochybení.

2.4.3 Kubernetes

Kubernetes je nejpoužívanější open-source platforma pro orchestraci kontejnerů. Předtím než byl Kubernetes zpřístupněn jako open-source, tak byl vyvíjen jako interní nástroj pro orchestraci kontejnerů ve společnosti Google. Své oblibě se Kubernetes těší díky široké škále funkcí, neustále rostoucímu ekosystému podpůrných nástrojů a rozsáhlé podpoře mezi poskytovateli cloudových služeb (Amazon Web Services, Google Cloud, Microsoft Azure a další). [23]

Výhody Kubernetes

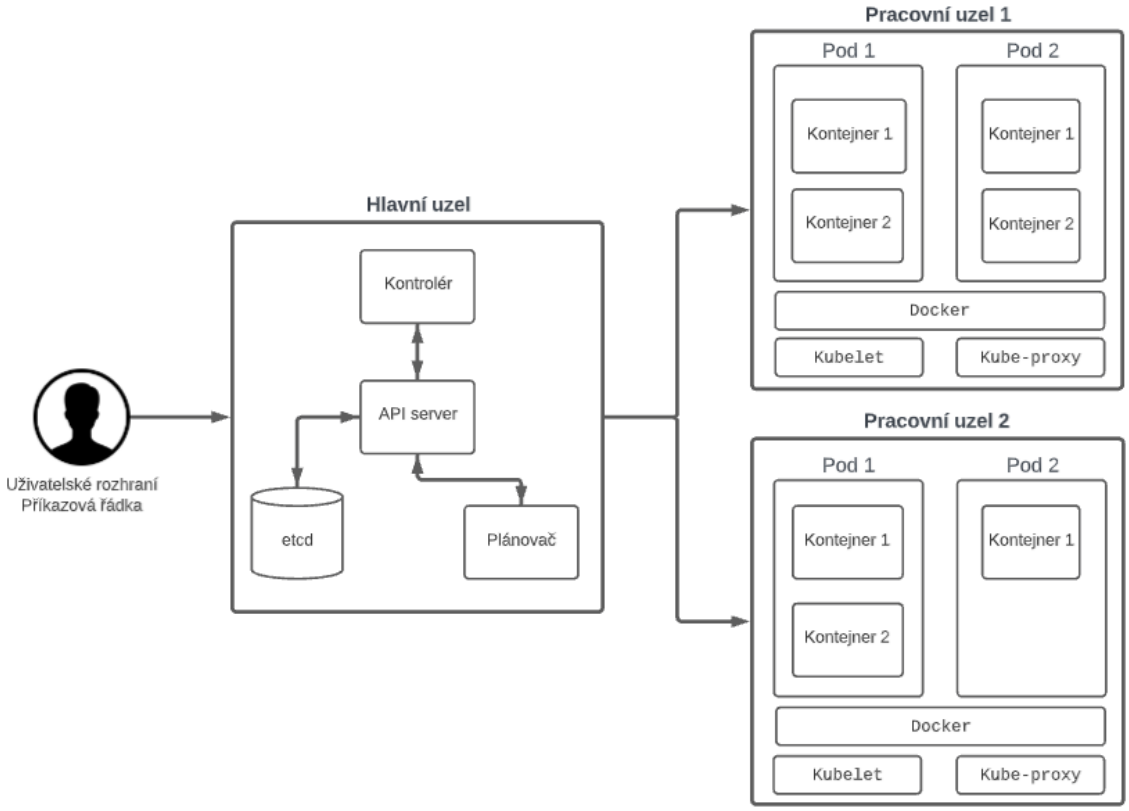
Výsledkem komplexnější a propracovanější funkčnosti v několika směrech jsou výhody, které na rozdíl od konkurenčních nástrojů Kubernetes nabízí. [23]

- Nasazení kontejnerů: Kubernetes nasadí kontejnery na určený hostitelský systém a udržuje je spuštěné v požadovaném stavu.
- Poskytování úložiště: Kubernetes podle potřeby dokáže kontejnery připojit k místnímu nebo cloudovému úložišti.
- Vystavení služby: Kubernetes umožňuje automaticky zpřístupnit kontejner jiným kontejnerům nebo internetu prostřednictvím IP adresy nebo DNS.
- Kontrola funkčnosti: Pokud se kontejner jakkoli poškodí nebo přestane fungovat, Kubernetes ho restartuje nebo nahradí.
- Vyrovnávání zátěže: Jestliže se zvýší provoz aplikace, Kubernetes ji automaticky škáluje a vyvažuje zátěž, aby byla zajištěna stabilita.

Architektura platformy Kubernetes

Kubernetes se řídí podle architektury klient-server a jeho hlavím cílem je zjednodušit správu kontejnerů poskytováním požadovaných funkcí přes REST API.

Obrázek 2.11 ukazuje architekturu Kubernetes klastru s jedním hlavním uzlem a dvěma pracovními uzly.



Obrázek 2.11: Architektura Kubernetes. Převzato a upraveno podle zdroje: [24]

- **Klastr**

Klastry jsou základními prvky architektury Kubernetes. Skládají se z uzlů, z nichž každý reprezentuje jeden hostitelský stroj (virtuální nebo fyzický). Každý klastr se skládá z uzlu hlavního (*Master node*), který funguje jako řídící uzel a kontaktní bod, a více pracovních uzlů (*Worker nodes*), které spouštějí a spravují kontejnerizované aplikace. Je možné mít klastr nastavený s více hlavními uzly, ale standardně se používá nastavení s jedním. Uživatelé mohou interagovat s klastrem pomocí uživatelského rozhraní nebo rozhraní příkazové řádky – **kubectl**, které komunikují přímo s řídícím uzlem přes API server. [24] [25]

- **Hlavní uzel**

Hlavní uzel obsahuje komponenty: [24] [25]

  - **etcd** – Jednoduchá distribuovaná databáze na principu klíč-hodnota pro ukládání informací o klastru (počet podů, jejich stav atd.). Z důvodu bezpečnosti je přístupná jen z API serveru.
  - **API server** – Přijímá všechny REST požadavky na úpravu klastru. Také jako jediná komponenta komunikující s etcd databází zajišťuje správné ukládání dat.
  - **Kontrolér** – Pomocí na pozadí běžících procesů reguluje stav klastru a provádí rutinní úlohy. Pokud se změní konfigurace (např. změna

parametru v konfiguračním souboru), kontrolér na nově nakonfigurovaném požadovaném stavu začne pracovat.
    - **Plánovač** – Na základě požadovaných zdrojů a dostupné výpočetní kapacity automatizuje, kdy a kde mají být kontejnery nasazeny. Plánovač musí mít informace o celkových dostupných zdrojích a o zdrojích aktuálně přidělených každému z uzlů.
- **Pracovní uzel**

Každý pracovní uzel obsahuje nástroj pro správu kontejnerů (Docker) a dvě další komponenty: [24] [25]

    - **Kubelet** – Pravidelně prostřednictvím API serveru přijímá nové nebo upravené nastavení podů a obstarává, že pody nebo jejich kontejnery nejsou poškozené a fungují v požadovaném stavu.
    - **Kube-proxy** – V případě potřeby vystavuje služby vnějšímu světu a provádí předávání požadavků do správných kontejnerů skrze izolované sítě uvnitř klastru.
- **Pod**

Pody jsou skupiny obsahující jeden či více kontejnerů, které by měly být řízeny jako jedna aplikace (sdílejí stejné výpočetní zdroje a síť). Když se provoz do kontejneru v podu zvětší nad hranici, kterou dokáže zvládnout, Kubernetes zreplikuje pod do dalších pracovních uzlů v klastru. [24] [25]

- **Nasazení (*Deployment*)**

Nasazení neboli deployment je soubor ve formátu YAML, který popisuje požadovaný stav podů (obrazy pro vytvoření kontejnerů, počet replik podů atd.). Kontrolér postupně upravuje prostředí tak, aby jeho aktuální stav odpovídal tomu specifikovanému v souboru nasazení. [24] [25]

2.5 Nextflow

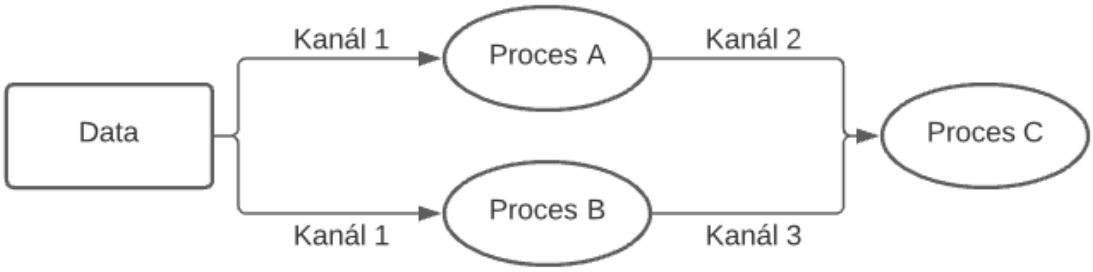
Nextflow je framework pro vytváření, spouštění a správu bioinformatických pipeline a zároveň se jedná o doménově specifický jazyk (DSL – *Domain Specific Language*), který vychází z programovacího jazyku pro platformu Java – Groovy. Jazyk Nextflow je navržen, aby měl minimální křivku učení. Noví uživatelé k vývoji pipeline využijí dosavadní dovednosti a nemusejí se tak učit nový programovací jazyk kompletně od začátku. [2] [26]

Kvůli neustále narůstajícím objemům dat jsou techniky pro jejich zpracování a analýzu čím dál tím více nezbytné. Nejlepším způsobem, jak se s problémem velkého množství dat vypořádat, je použití paralelizovaných a distribuovaných výpočtů. Bohužel běžně dostupné bioinformatické nástroje tyto možnosti neposkytují a pokud ano, tak jen velice omezeně, ale použití těchto nástrojů vyžaduje komplexní znalosti. Základem

frameworku Nextflow je dataflow programovací model, který značně simplifikuje psaní paralelních a distribuovaných pipeline bez nadbytečné složitosti a s cílem soustředit se na tok dat, tedy na funkční logiku aplikace. Nextflow je navržen na základě platformy Linux, která poskytuje velké množství jednoduchých, ale výkonných nástrojů pro příkazový řádek a skriptování. Tyto nástroje pohromadě usnadňují složité manipulování s různými druhy dat. Vypadá to tak, že skript Nextflow je složen z několika procesů. Každý proces spouští daný skript nebo nástroj a k tomu je přidána možnost koordinace a synchronizace spouštění procesů pouhým specifikováním jejich vstupů a výstupů. [2] [26]

2.5.1 Struktura pipeline

Pipeline mohou být reprezentovány jako grafy, kde uzly jsou procesy a hrany jsou kanály. Procesy jsou bloky kódu napsané v jakémkoliv programovacím jazyce spustitelném na platformě Linux (Python, Bash, Perl atd.), které mohou být spouštěny stejně jako skripty nebo programy, kdežto kanály jsou asynchronní fronty spojující jednotlivé procesy pomocí vstupů a výstupů. Každý proces je nezávislý vůči ostatním a může být spouštěn paralelně v závislosti na počtu prvků v kanálu. Z obrázku 2.12 zobrazujícího strukturu pipeline je patrné, že procesy A a B mohou běžet paralelně, ale proces C je spuštěn teprve až po jejich dokončení. [26]

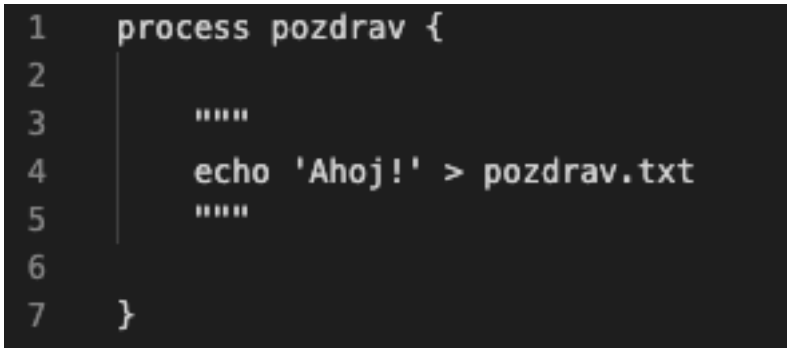


Obrázek 2.12: Struktura pipeline. Převzato a upraveno podle zdroje: [26]

2.5.2 Procesy

Proces označuje základní procesní primitiv pro spouštění uživatelských skriptů. Definice procesu musí začínat klíčovým slovem `process`, za kterým následuje název a tělo procesu ohraničeno složenými závorkami. Tělo musí obsahovat textový řetězec reprezentující procesem spouštěný příkaz nebo skript. [27]

Příklad jednoduchého procesu je znázorněn na obrázku 2.13.



Obrázek 2.13: Příklad procesu. Zdroj: vlastní

Uvnitř každého procesu může být definováno až 5 bloků: direktivy, vstupy, výstupy, podmínkovou klauzuli a skript. Syntax procesu je definována na obrázku 2.14.

Obrázek 2.14: Syntax procesu. Převzato a upraveno podle zdroje: [27]

Direktivy

Použitím bloku direktiv lze poskytnout aktuálnímu procesu volitelné nastavení, a tím ovlivnit jeho průběh. Direktivy musí být definovány v horní části těla procesu před kterýmkoli jiným blokem. Některá nastavení jsou dostupná pro všechny procesy, ale některá závisí na právě používané exekuční platformě. [27]

Vstupy

Jelikož jsou jednotlivé procesy od sebe navzájem izolované, jedinou možností, jak mohou mezi sebou komunikovat, je posílání hodnot pomocí kanálů. Blok vstupů určuje, z jakých kanálů má proces přijímat data. Je možné definovat pouze jediný blok vstupů a ten musí deklarovat jeden nebo více vstupních kanálů. [27]

Výstupy

Blok výstupů umožňuje uživateli definovat kanály používané procesem k odesílání vytvořených výsledků. V procesu může existovat pouze jeden blok výstupů, který musí obsahovat jeden nebo více výstupních kanálů. [27]

Podmínka

Část deklarovaná výrazem `when` dovoluje uživateli definovat podmínku, jež musí být splněna, aby mohl být proces spuštěn. Podmínkou může být jakýkoliv výraz nabývající pouze dvou hodnot – pravda (*true*) respektive nepravda (*false*). Využití podmínky je užitečné, pokud je potřeba povolit či zakázat spouštění procesu na základě různých hodnot vstupů a parametrů. [27]

Skript

Proces může obsahovat pouze jediný blok kódu definující spustitelný příkaz nebo skript jako textový řetězec (*string*). Pokud proces obsahuje deklarace vstupů a výstupů, musí být navíc blokem posledním. Zadaný řetězec se spustí jako skript v jazyce Bash a může se skládat z jakéhokoliv příkazu, skriptu anebo jejich kombinace, které by normálně šly spustit v příkazovém řádku nebo jako běžný Bash skript. Jediným omezením pro použití příkazů je, že příkazy musí být dostupné v systému, kde je Nextflow skript spouštěn. [27]

Tělo skriptu obsahující kód disponuje více možnostmi, jak jej zapsat. Textový řetězec lze definovat pomocí jednoduchých uvozovek (`'`) nebo dvojitých uvozovek (`"`) a více řádkové řetězce třemi znaky jednoduchých (`'''`) nebo dvojitých uvozovek (`"""`). Mezi těmito způsoby je nepatrný ale velice podstatný rozdíl. Řetězce začínající znakem dvojitých uvozovek podporují substituci proměnných, zatímco řetězce začínající jednoduchými uvozovkami takovou funkcionalitu nepodporují. [27]

Problém nastává, pokud je potřeba v kódu přistupovat jak k proměnným Nextflow tak k systémovým proměnným. Oba typy se totiž označují znakem dolaru (`$`) před samotným názvem. Naštěstí jde situaci vyřešit prostřednictvím tzv. „escapování“ systémových proměnných, což znamená, že se před systémové proměnné přidá znak zpětného lomítka (`\`). [27]

Proměnná `$pozdrav` z ukázky kódu na obrázku 2.15 musí být definována ve skriptu dříve před procesem a Nextflow ji poté nahradí skutečnou hodnotou. Oproti tomu proměnná `$USER` musí existovat v prostředí systému, kde je skript spouštěn.

```
1   pozdrav = 'ahoj'
2   process prikladEscaping {
3
4       script:
5       """
6       echo Uzivatel \$USER rika $pozdrav
7       """
8
9   }
```

Obrázek 2.15: Příklad „escapování“ systémové proměnné. Zdroj: vlastní

Shell

Alternativou k bloku definovanému jako `script` je blok `shell`. Rozlišuje je ale jeden důležitý rozdíl. Shell používá znak vykřičníku (`!`) jako označení pro Nextflow proměnně místo obvyklého dolaru (`$`). Takto je možné použít Nextflow i Bash proměnné ve stejném kódu (viz obrázek 2.9) bez nutnosti escapování Bash proměnných jako je to v případě `script` bloku. Kód procesu se tak stává lépe čitelnějším a udržovatelným. [27]

V příkladu na obrázku 2.16 je znázorněno použití obou typů proměnných v jednom kusu kódu. Proměnná `$USER` je spravována systémovým interpretem Bash, ale `!{ovoce}` je zpracována pomocí Nextflow jako proměnná v rámci procesu.

```
1   process shellPriklad {
2
3       input:
4       val ovoce from 'jablko', 'banan', 'mango'
5
6       shell:
7       '''
8       echo Uzivatel $USER ma rad !{ovoce}
9       '''
10
11  }
```

Obrázek 2.16: Příklad použití bloku `shell`. Zdroj: vlastní

Nativní kód

V procesech je možné spouštět i nativní kód jiný než pouze systémové skripty. To znamená, že místo specifikování příkazů v textovém řetězci lze proces definovat pomocí jazyka Nextflow jako ve zbytku skriptu. K docílení této funkcionality stačí v procesu zaměnit klíčové slovo `script` za výraz `exec`. [27] Příklad použití nativní exekuce kódu je na obrázku 2.17.

```
1  process vypisPozdravy {
2
3      input:
4      val pozdrav from 'Ahoj', 'Hello', 'Ciao'
5
6      exec:
7      println "$pozdrav"
8
9  }
```

Obrázek 2.17: Nativní exekuce. Zdroj: vlastní

2.5.3 Kanály

Jelikož je framework Nextflow vytvořen na základě programovacího modelu Dataflow, tak jediný způsob, jak mohou mezi sebou jednotlivé procesy komunikovat je prostřednictvím kanálů. Kanály se vyznačují dvěma zásadními vlastnostmi. Zaprvé, odesílání dat je asynchronní operace, která se dokončí okamžitě bez toho, aby musela čekat na proces, jenž data přijímá. Druhou vlastností je, že příjem dat blokuje celý proces, dokud všechna data nedorazí. [28]

Nextflow rozlišuje mezi dvěma typy kanálů: kanál fronty (*queue channel*) a kanál hodnoty (*value channel*).

Kanál fronty (*Queue channel*)

Jedná se o neblokující jednosměrnou FIFO (First In, First Out) frontu spojující dva procesy. Kanály fronty bývají většinou vytvářeny některou z předdefinovaných metod (`of`, `from`, `fromList`) nebo jako výstup procesu použitím výrazu `into`. Stejný kanál nemůže být použit vícekrát než jednou jako vstup procesu a jednou jako výstup procesu. Pokud je potřeba propojit kanál s více než jedním procesem, lze vytvořit několik jeho kopií a každou použít zvlášť pro jednotlivé procesy. [28]

Kanál hodnoty (*Value channel*)

Také označovaný jako singleton kanál je vázán na jedinou hodnotu a je možné ho číst neomezeně krát. Z tohoto důvodu může kanál propojovat více procesů najednou.

Kanál lze vytvořit pomocí předdefinované metody `value` nebo pomocí operátorů vracející pouze jedinou hodnotu (`min`, `max`, `sum`, `count` atd.). [28]

2.5.4 Nextflow a kontejnery

Nextflow podporuje několik různých platforem pro kontejnerizaci, což umožňuje zabalením knihoven a závislostí do jednotného formátu vytvářet samostatné a reprodukovatelné pipeline.

Docker

Nextflow skript není potřeba nijak upravovat, aby jej bylo možné spustit pomocí Dockeru. Stačí definovat Docker obraz, ze kterého mají být kontejnery vytvořeny. Při každém spuštění skriptu ho Nextflow spustí uvnitř vytvořeného kontejneru. Prakticky to vypadá tak, že Nextflow automaticky zabalí všechny procesy skriptu a spustí je pomocí příkazu `docker run` spolu se zadaným Docker obrazem. [29]

Součástí poskytnutých Docker obrazů může být jakýkoliv nástroj potřebný k provedení procesu. Kontejnery jsou spouštěny způsobem, který zaručuje, že soubory s výsledky procesů se vytváří v hostitelském souborovém systému. Díky tomu nejsou pro spuštění nutné žádné další kroky a nijak to neovlivňuje průběh pipeline. [29]

Singularity

Integrace s platformou Singularity je založena na stejném modelu pro spouštění kontejnerů jako v případě Dockeru. Platí i stejná pravidla, není potřeba skript upravovat a stačí jen poskytnout obrazový soubor Singularity. Výhodou Singularity je možnost spouštění kontejnerů bez nejvyššího oprávnění a také, že nevyužívá samostatný proces daemona. Proto je Singularity vhodnější pro použití s technologií HPC. [29]

2.5.5 Exekuční abstrakce

Zatímco v architektuře Nextflow proces určuje kód (příkaz nebo skript), který musí být spuštěn, exekutor definuje, jakým způsobem je v cílovém systému proces skutečně proveden. Nextflow tudíž vytváří abstrakční vrstvu mezi funkční logikou pipeline a systémem, na kterém je pipeline spouštěna. Každá pipeline může být díky tomu bez problému spuštěna na lokálním počítači, gridové platformě nebo v cloudu.

Níže je uvedeno několik podporovaných exekučních platforem:

Lokální

Ve výchozím nastavení se používá lokální počítač, který je velmi užitečný při testování a vývoji pipeline. Lokální exekutor využívá předností vícejádrové architektury poskytovaného procesoru a jednotlivé procesy paralelizuje vytvářením více vláken. [30]

Kubernetes

Nextflow disponuje vestavěnou podporou pro platformu Kubernetes, která prostřednictvím Kubernetes klastru zefektivňuje spouštění kontejnerizovaných pipeline. Exekutor Kubernetes spouští každou pipeline jako objekt Pod, jenž provádí potřebnou orchestraci pro úspěšné dokončení pipeline. [30]

AWS Batch

AWS Batch je výpočetní služba umožňující spouštění kontejnerizovaných (používá Docker) úloh v rámci cloudové infrastruktury Amazon. Flexibilně přiřazuje optimální množství výpočetních zdrojů na základě požadavků spouštěných úloh. Nextflow poskytuje podporu pro AWS Batch, která dovoluje snadné nasazení pipeline do cloudového prostředí bez nutnosti spravování klastru virtuálních strojů. [30]

Slurm

Slurm je open-source systém pro správu klastru a plánovač úloh. Poskytuje vlastní framework pro spouštění a monitorování paralelních úloh a řízením fronty čekajících úloh rozhoduje o přidělení zdrojů. Nextflow odesílá do Slurm klastru každý proces jako samostatnou výpočetní úlohu. [31]

2.5.6 Sdílení pipeline

Součástí Nextflow je i integrace s hostovanými platformami pro ukládání a sdílení zdrojových kódů, jako GitHub, GitLab nebo BitBucket. Díky této funkcionalitě je možné konzistentněji spravovat vyvíjený kód nebo používat pipeline jiných uživatelů rychlým a jednoduchým způsobem.

Při spuštění pipeline se Nextflow nejprve v rámci prostředí snaží najít soubor se zadaným názvem, ale jestliže neexistuje, začne hledat na platformě GitHub (pokud není jinak nastaveno) veřejné úložiště se shodným názvem. V případě, že je repozitář nalezen, tak se automaticky stáhne a pipeline se spustí. Pro správné použití pipeline ze vzdáleného úložiště je nutné dodržet korektní syntax. Název musí být ve tvaru – `{uzivatelsko_jmeno}/{nazev_repozitare}`. [32]

2.5.7 Výhody Nextflow

Rychlé prototypování

Umožňuje rychlé vytvoření malé pipeline, která může být vyvíjena inkrementálně. Každá úloha je nezávislá a může být jednoduše přidána k dalším. Možností je i přepoužívání již existujících skriptů a nástrojů bez nutnosti jejich upravování a bez závislosti na programovacím jazyce. [33]

Reprodukovatelnost

Podporuje využití kontejnerových technologií Docker a Singularity. Jejich využití dělá pipeline reprodukovatelné v jakémkoliv prostředí UNIX. Navíc integrace s platformami pro sdílení a ukládání zdrojových kódů umožňuje přímé použití specifické verze pipeline přímo z repozitáře. Nextflow ji stáhne a použije přímo za běhu. [33]

Přenositelnost

Nextflow lze použít na spoustě platforem bez nutnosti upravování kódu. Podporuje několik plánovačů úloh (Sun Grid Engine, Slurm, Portable Batch System atd.) a cloudové platformy jako Amazon Web Services, Google Cloud Platform a další. [33]

Škálovatelnost

Základem Nexflow je programovací model Dataflow, který zjednodušuje vytváření složitých pipeline. Nextflow se postará o paralelizování procesů bez dalšího dodatečného kódu. Výsledné aplikace jsou paralelizované a lze je škálovat bez přizpůsobení specifické platformní architektuře. [33]

Obnovitelnost

Všechny průběžné výsledky produkované v rámci spuštěné pipeline jsou automaticky sledovány. Pro každý proces se vytvoří dočasná složka, a ta je v případě potřeby, při opakovaném spuštění pipeline, obnovena. [33]

3 Cíle práce

Hlavním cílem práce je vytvoření univerzálního nástroje pro spouštění, správu a monitorování bioinformatických pipeline. Motivací pro jeho vytvoření je snaha usnadnit vývoj a rozvoj laboratorních informačních systémů, převážně v oblasti automatizace každodenních výpočetních úkonů. Takový systém musí být vysoce integrovatelný pomocí standardních nástrojů a umožnit rychlý vývoj existujících aplikací. Pro splnění výše uvedeného byly definovány konkrétní podcíle této práce:

- Popsat možnosti nástroje Nextflow pro exekuci bioinformatických pipeline včetne nasazení.
- Implementovat či integrovat existující REST API pro ovládání Nextflow a nástroje, jako celku.
- Implementovat či integrovat existující řešení pro monitoring pipeline.
- Vytvořit dokumentaci k nástroji a instalační příručku pro uvedení nástroje do provozu.
- Zmapovat a popsat možnosti kontroly zdrojů (resource management) pro tento nástroj.

4 Metody

V následující kapitole jsou shrnuty podrobnosti o návrhu, použitých technologiích a implementaci vyvíjeného nástroje.

Rešerší bylo zjištěno, že vývojem podobného produktu [34] se zabývali vědci z univerzity Clemson v Jižní Karolíně, konkrétně Dr. Ben Sherman. S vývojem ale v roce 2021 skončil, a protože se jedná o open-source projekt, mohl být použit jako základní stavební kámen pro nástroj, který je hlavním předmětem této práce.

Zdrojový kód vytvořeného nástroje je volně šiřitelný pod licencí MIT.

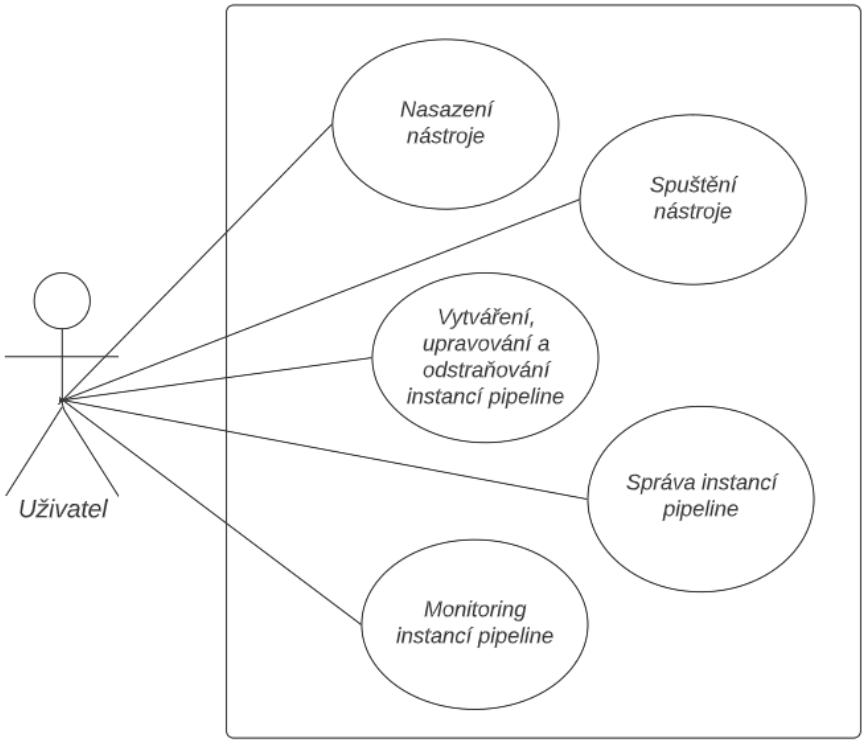
4.1 Analýza požadavků

Jednotlivé požadavky vychází ze zadání a cílů práce.

- REST API, které umožní instance bioinformatických pipeline:
  - Vytvářet
  - Spouštět
  - Zastavovat
  - Znovu spouštět
  - Upravovat
  - Monitorovat
  - Odstraňovat
- Snadné nasazení a uvedení do provozu
- Automatické přiřazování a správa výpočetních zdrojů
- Možnost monitorování výpočetních zdrojů pro každý běh
- Umožnit přiřazování priority jednotlivým pipeline (pipeline A má vyšší prioritu než pipeline B, proto bude spuštěna dřív)

4.2 Případy užití

Jelikož použití nástroje vyžaduje alespoň minimální znalosti programování, které jsou dostačující i k jeho zprovoznění, a navíc požadavky nejsou specifikovány pro více uživatelských rolí, jsou případy užití (viz obrázek 4.1) definovány pro jediného aktéra – uživatele.



Obrázek 4.1: Diagram případů užití. Zdroj: vlastní

Nasazení nástroje

Nástroj musí být možné nasadit jednoduchým způsobem do různých prostředí na několika platformách.

Spuštění nástroje

Uvedení nástroje do provozu nesmí vyžadovat žádné speciální dovednosti a musí být proveditelné co nejmenším počtem příkazů.

Vytváření, upravování a odstraňování instancí pipeline

Uživatel má možnost vytvářet, po vytvoření upravovat a odstraňovat jednotlivé instance pipeline.

Správa instancí pipeline

Správa nad jednotlivými instancemi pipeline zahrnuje operace jako je jejich spouštění, zastavování, restartování.

Monitoring instancí pipeline

Monitoring umožňuje sledovat akruální stav instancí pipeline, jako například čekající, běžící, dokončená a také informace o jejich průběhu neboli log.

4.3 Výběr technologií

Použité technologie byly vybrány podle analýzy běžně dostupných a nejčastěji používaných nástrojů z první poloviny práce. Zároveň byl kladen důraz na to, aby pomocí moderních trendů splňovali stanovené požadavky a byly jednoduché na implementaci. Některé technologie byly předem určeny zadáním práce nebo výběrem již existujících projektů jako základ nástroje.

4.3.1 Nextflow

Framework Nextflow již sám o sobě splňuje některé z kladených požadavků a nabízí prostředky pro práci se standardizovanými pipeline. Obrovskou výhodou Nextflow je, že poskytuje celkem snadné integrování velkého množství moderních nástrojů a tato podpora se s každou další vydanou verzí rozšiřuje o nové možnosti.

4.3.2 Python

Python je open-source vysokoúrovňový programovací jazyk. Je zařazován mezi skriptovací jazyky, ale svými možnostmi tuto kategorii značně převyšuje. Python podporuje různé způsoby programování, jako objektově orientovaný přístup, procedurální programování nebo funkcionální programování. Způsoby je možné používat volitelně, a to i v rámci jedné aplikace, záleží jen na schopnostech programátora a řešeném problému. Díky zmíněné variabilitě a velmi dobré čitelnosti je jazyk Python velice efektivní a produktivní z pohledu rychlosti vytváření aplikací. K dispozici je veliké množství knihoven, které značně usnadňují řešení problémů z různých oblastí, a tím významně zrychlují vývoj aplikací, od těch jednoduchým se stručným zdrojovým kódem až po komplexní skládající se z několika modulů. [35]

Framework Tornado

Tornado je webový framework pro programovací jazyk Python vytvořený speciálně pro práci s asynchronními procesy. Je navržen, aby byl jednoduchý a škálovatelný. Nejčastěji je používaný pro webové aplikace, které vyžadují dobré škálování, ale lze jej použít i v mnoho dalších případech. Klíčovými vlastnostmi frameworku jsou: [36]

- Schopnost zvládnout až tisíce připojení za sekundu
- Jednoduchý způsob vytváření API
- Předpřipravené komponenty

Díky těmto vlastnostem a podpoře neblokujících webových soketů je ideálním řešením pro aplikace, které delší dobu čekají na síťové požadavky nebo generování výstupu. Pro snadnější vytváření složitějších částí webových aplikací je součástí frameworku sada knihoven. Na rozdíl od velkých frameworků poskytuje Tornado jen vážně potřebné knihovny, což udržuje kód čistším.

4.3.3 MongoDB

MongoDB je jednoduchá, objektově orientovaná a škálovatelná open-source databáze. Oproti tradičním relačním databázím používá kolekce místo tabulek a místo řádků dokumenty. Veškerá data jsou ukládána ve formátu JSON a využívá se dynamické databázové schéma, to znamená, že data nemusejí mít předem daný formát. MongoDB umožňuje aplikacím rychlou a jednoduchou integraci dat. [37]

Jelikož se počítá s tím, že všechna data budou ve formátu JSON, je MongoDB skvělou možností pro ukládání dat k jednotlivým instancím pipeline.

4.3.4 Docker

Kontejnerizace je velice důležitým krokem při vývoji nástroje. Izolování nástroje do kontejneru zajišťuje jeho snadnou přenositelnost a signifikantně zjednodušuje nasazení do jakéhokoliv prostředí. Platforma Docker je volbou číslo jedna, protože se v dnešní době považuje za standard v technologii kontejnerů a nabízí programátorům přívětivé prostředí s množstvím užitečných funckionalit. Zabalení nástroje do kontejneru navíc umožňuje použití platforem pro orchestraci, a tím automaticky řídit a monitorovat přiřazování výpočetních zdrojů.

4.3.5 Kubernetes

Pro splnění požadavků na práci s výpočetními zdroji, jako je jejich monitorování, automatické přiřazování a plánování je potřeba využít nástroj pro orchestraci. Kubernetes je nejrozšířenější platformou mezi uživateli a také nejdostupnější mezi provozovateli cloudových služeb. Poskytuje širokou paletu funkcí, které se dají okamžitě použít a ušetří tak čas. Součástí Nextflow je zabudovaná podpora pro platformu Kubernetes, je možné ji použít jako exekutor pro pipeline. Proto je ideálním spojením mezi Nextflow a technologií kontejnerů, v tomto případě Dockerem.

Kubernetes dashboard

Dashboard je webové uživatelské rozhraní pro správu a monitoring Kubernetes klastru. Lze jej využít pro odstraňování problémů s kontejnerizovanou aplikací a správu výpočetních prostředků. Dashboard poskytuje přehled o stavu běžících aplikací v klastru a informace o stavu zdrojů či případných problémech, které se mohly vyskytnout. [38]

Minikube

Minikube je nástroj, pomocí kterého je možné snadno a rychle vytvořit lokální Kubernetes klastr v systémech Linux, Windows a macOS. Minikube je velice užitečným při vývoji na lokálním počítači, protože pokrývá většinu funkcí dostupných v normálním Kubernetes klastru. [39]

4.3.6 Helm

Zatímco kontejnerizace zlepšuje reprodukovatelnost a zjednodušuje nasazení aplikací, Kubernetes vnáší do těchto akcí více komplexity. Helm pomáhá nasazovat a spravovat Kubernetes aplikace prostřednictvím souborů ve formátu YAML, tzv. schémat (*charts*). Schéma (*Chart*) představuje zabalenou aplikaci, je to soubor předkonfigurovaných prostředků aplikace, které lze nasadit jako jeden celek. Způsob, jakým Helm dokáže spravovat jednotlivé aplikace, zvyšuje produktivitu a snižuje složitost nasazení. Často pro nasazení aplikace stačí jediný příkaz – `helm install`. [40]

4.3.7 GitHub

GitHub je webová platforma poskytující verzovací systém pro uchovávání, sdílení a správu zdrojových kódů. Pro uložení kódu projektu diplomové práce je GitHub vhodný zejména díky zabudovanému nástroji GitHub Actions, který dokáže automatizovat nasazení a urychlit vývoj aplikace. [41]

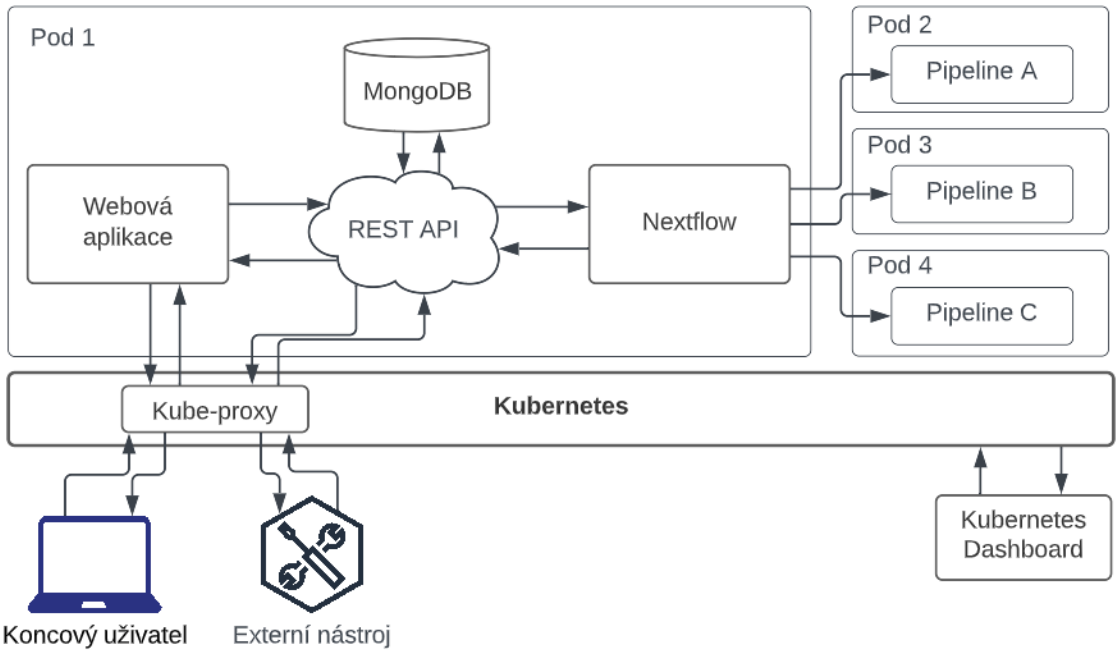
4.3.8 AngularJS

AngularJS je javascriptový framework pro vytváření webových aplikací, nejčastěji jednostránkových (*single-page*). [42] V práci je použit, protože její součástí je i ukázka webového rozhraní pro koncové uživatele.

4.4 Architektura řešení

Konečná architektura nástroje vznikla na základě použitých technologií a dílčích architektonických pravidel. Každá komponenta hraje důležitou roli ve fungování nástroje jako celku a byla vybrána i na základě její vnitřní architektury z toho důvodu, aby bylo její integrování do výsledného produktu co nejsnazší.

Obrázek 4.2 přibližuje zjednodušenou architekturu výsledného řešení.



Obrázek 4.2: Simplifikovaná architektura výsledného nástroje. Zdroj: vlastní

REST API

REST API je ústřední komponentou spojující všechny části nástroje dohromady. Může přijímat požadavky buď přímo přes koncové body nebo prostřednictvím uživatelského rozhraní, které jen zjednodušuje a uživateli zpřehledňuje přímé využití koncových bodů. Po zpracování požadavku komponenta uloží potřebné informace do databáze nebo zpracovaný požadavek předá frameworku Nextflow, jenž provede nezbytné operace pro úspěšné dokončení úlohy. Například při přijetí požadavku na spuštění instance pipeline aktualizuje v jejím databázovém dokumentu aktuální stav (na běžící) a ve stejné chvíli příkazem frameworku Nextflow zahájí spuštění instance.

Nextflow

Nextflow má na starost veškeré pracovní úkony týkající se bioinformatických pipeline. Uživatel je od komponenty odstíněn skrze REST API, a tudíž s ní nepřijde napřímo do styku. REST API definuje množinu možných operací nad instancemi pipeline a pokud by chtěl uživatel použít funkcionalitu, kterou Nextflow disponuje, ale REST API takovou možnost neposkytuje, musel by tak učinit jedině změnou zdrojového kódu a opětovným nasazením nástroje.

Kubernetes

Platforma Kubernetes má na starost orchestraci samotného nástroje a také má funkci exekutoru Nextflow skriptů. Pokaždé, když má být instance pipeline spuštěna, tak je zabalena do vlastního kontejneru a pro běh kontejneru je vytvořen samostatný pod. Nový pod je vždy po dokončení pipeline odstraněn. Manipulace s pody tímto způsobem umožňuje řízení a správu výpočetních prostředků pro každou pipeline.

Aby bylo vůbec možné nástroj používat, je k tomu potřeba služby Kube-proxy. Ta hraje důležitou roli v komunikaci kontejnerů s vnějším světem, kdy vystavuje služby REST API a webové aplikace do vnější sítě a zprostředkovává spojení až ke kontejnerům běžících uvnitř Kubernetes podu. Koncový uživatel nebo externí nástroj mohou následně bez problémů využít webovou aplikaci, popřípadě služby REST API.

Platforma Kubernetes nabízí jednu velice užitečnou a pro tento nástroj důležitou funkcionalitu. Jde o možnost přiřazení priority jednotlivým podům. A jelikož se každá instance pipeline spouští ve svém samostatném podu, je to jednoduchý způsob, jak vyřešit požadavek na jejich prioritizaci.

MongoDB

Databáze MongoDB funguje jako úložiště pro veškeré informace o instancích pipeline. Je využívaná komponentou REST API, obousměrná komunikace probíhá prostřednictvím knihovny `PyMongo`, která usnadňuje zápis dotazů pro ukládání nebo získávání dat.

Webová aplikace

Webová aplikace slouží jako ukázka uživatelského rozhraní pro koncové uživatele. Ukazuje, jak by v případě použití nástroje, mohly externí nástroje používat koncové body komponenty REST API pro práci s instancemi pipeline. Zdrojový kód aplikace je vytvořen v programovacím jazyce AngularJS.

4.5 Nasazení

K nasazení nástroje je v první řadě potřeba mít vytvořený Kubernetes klastr a nástroj pro příkazovou řádku (`kubectl`) nakonfigurovaný ke komunikaci s tímto klastrem. V této práci je k ukázkovému nasazení použit lokální Kubernetes klastr vytvořený nástrojem `minikube`. K vytvoření klastru stačí jediný příkaz:

- `minikube start`

Výstup po použití příkazu by měl vypadat přibližně jako na obrázku 4.3.

```
~ minikube start
😄  minikube v1.24.0 on Darwin 10.15.7
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🏃  Updating the running docker "minikube" container ...
🐳  Preparing Kubernetes v1.22.3 on Docker 20.10.8 ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
    ▪ Using image kubernetesui/dashboard:v2.3.1
    ▪ Using image kubernetesui/metrics-scraper:v1.0.7
🌟  Enabled addons: storage-provisioner, dashboard

❗  /usr/local/bin/kubectl is version 1.19.7, which may have incompatibilites with Kubernetes 1.22.3.
    ▪ Want kubectl v1.22.3? Try 'minikube kubectl -- get pods -A'
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Obrázek 4.3: Vytvoření lokálního Kubernetes klastru příkazem `minikube start`. Zdroj: vlastní

Další krokem je povolení doplňku pro sběr dat o využití výpočetních zdrojů. Je k tomu určen příkaz:

- `minikube addons enable metrics-server`

Jeho výstup je znázorněn na obrázku 4.4.

```
[→  ~ minikube addons enable metrics-server
    ▪ Using image k8s.gcr.io/metrics-server/metrics-server:v0.4.2
🌟  The 'metrics-server' addon is enabled
```

Obrázek 4.4: Povolení doplňku pro sběr dat ohledně využití výpočetních zdrojů. Zdroj: vlastní

Kubernetes klastr je tak připravený a teď je na řadě stažení repozitáře se zdrojovým kódem nástroje. Kód se nachází na adrese – `https://github.com/ihitch/DP.git` – a s jeho stažením prostřednictvím příkazu `git clone` pomůže verzovací nástroj `git` (viz obrázek 4.5).

```
[→  ~ git clone https://github.com/ihitch/DP.git
Cloning into 'DP'...
remote: Enumerating objects: 52, done.
remote: Counting objects: 100% (52/52), done.
remote: Compressing objects: 100% (45/45), done.
remote: Total 52 (delta 2), reused 49 (delta 2), pack-reused 0
Unpacking objects: 100% (52/52), done.
```

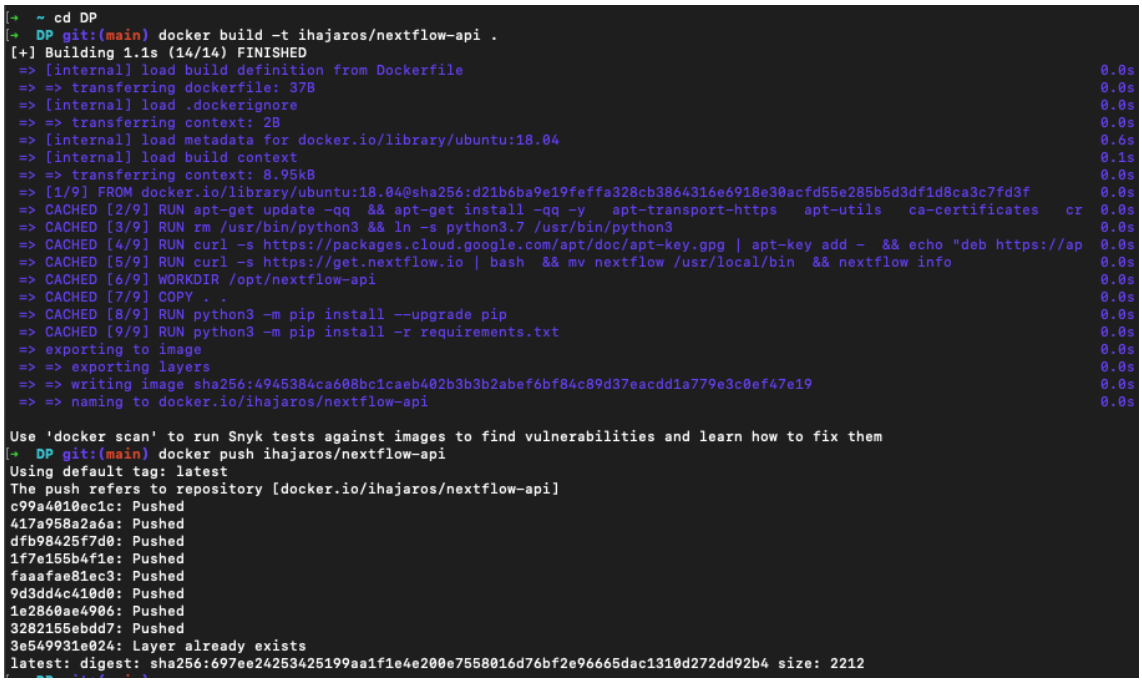
Obrázek 4.5: Stažení repozitáře. Zdroj: vlastní

Po stažení repozitáře je potřeba změnit název Docker obrazu v souboru `values.yaml`, který se nachází ve složce `helm`. Název obrazu je upřesněn v nastavení služby `WebServer` pod klíčem `Image` (viz obrázek 4.6) a jeho doporučený formát je `<uzivatelske_jmeno>/<nazev_obrazu>`. Stejný název musí být použit ze složky projektu při vytvoření Docker obrazu a jeho nahrání do vzdáleného registru (ukázka příkazů na obrázku 4.7):

- `cd DP`
- `docker build -t <uzivatelske_jmeno>/<nazev_obrazu> .`
- `docker push <uzivatelske_jmeno>/<nazev_obrazu>`

```
20  # Web server deployment settings
21  WebServer:
22    # Docker image, change to your nextflow-api image if needed
23    Image: ihajaros/nextflow-api
```

Obrázek 4.6: Změna názvu Docker obrazu pro nasazení. Zdroj: vlastní



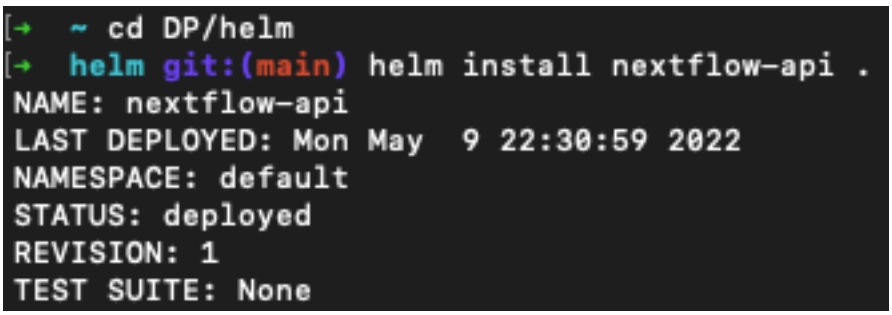
Obrázek 4.7: Vytvoření Docker obrazu a jeho nahrání do registru. Zdroj: vlastní

Následuje změna pracovního adresáře:

- `cd helm`

Teď je možné aplikaci jednoduše nasadit jediným příkazem (příklad na obrázku 4.8):

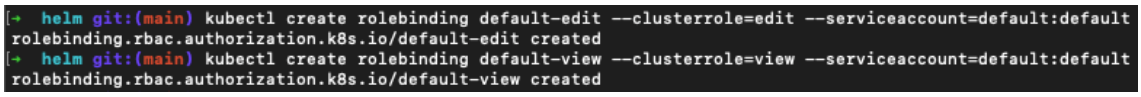
- `helm install nextflow-api .`



Obrázek 4.8: Nasazení nástroje. Zdroj: vlastní

Ještě je potřeba dát frameworku Nextflow potřebná práva, aby mohl vytvářet pody v Kubernetes klastru a monitorovat výpočetní zdroje. Jde o dva příkazy a jejich použití je zobrazeno na obrázku 4.9:

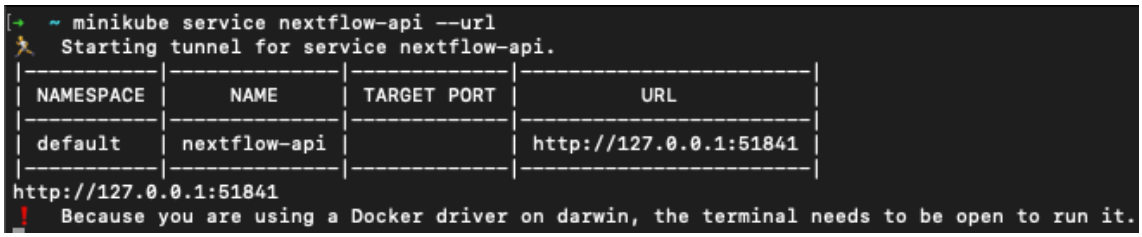
- `kubectl create rolebinding default-edit --clusterrole=edit --serviceaccount=default:default`
- `kubectl create rolebinding default-view --clusterrole=view --serviceaccount=default:default`



Obrázek 4.9: Vytvoření práv pro Nextflow. Zdroj: vlastní

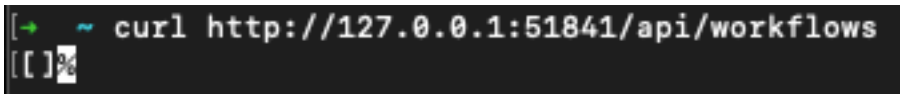
Nástroj je nasazený a stačí jen dát Kubernetes klastru pokyn pro vystavení jeho služeb vnějšímu světu (obrázek 4.10). K tomu je určen příkaz:

- `minikube service nextflow-api --url`



Obrázek 4.10: Vystavení nástroje. Zdroj: vlastní

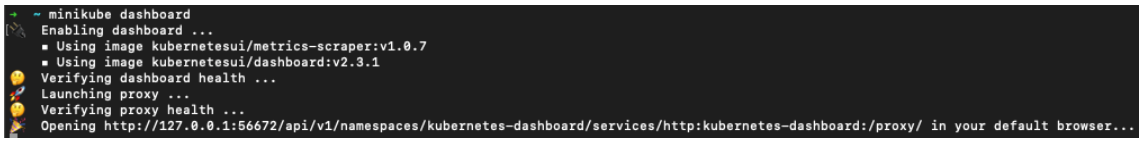
Nástroj je nyní dostupný na vypsané adrese. Funkčnost jde ověřit dvěma způsoby, buď otevřením adresy v prohlížeči a zobrazí se uživatelské rozhraní nebo odesláním požadavku na některý z koncových bodů REST API. Na obrázku 4.11 je příklad požadavku GET (nástroj `curl`) na koncový bod pro výpis všech instancí pipeline. Odpověď obsahuje prázdné pole, jelikož ještě nebyla žádná instance vytvořena.



Obrázek 4.11: Výpis všech instancí pipeline. Zdroj: vlastní

Uživatel má také k dispozici webové rozhraní pro monitorování stavu a správu Kubernetes klastru včetně využití výpočetních zdrojů. Kubernetes Dashboard se otevře ve výchozím prohlížeči (obrázek 4.12) po spuštění příkazu:

- `minikube dashboard`



Obrázek 4.12: Spuštění služby Kubernetes Dashboard. Zdroj: vlastní

4.5.1 Konfigurace služeb

Díky použití nástroje `Helm` je poměrně snadné konfigurovat některé z parametrů či konkrétní nastavení nástroje. Všechny upravitelné hodnoty jsou shromážděny v souboru `DP/helm/values.yaml` a jejich konfigurací před nasazením je možné měnit například počet procesorů, paměť RAM nebo velikost úložiště pro běh aplikace.

Uživatel má také možnost upravovat stávající nebo vytvářet nové prioritní třídy. Aktuálně používané třídy jsou definované v souboru `DP/helm/templates/pc.yaml`. Jedná se o 4 třídy:

- **high-priority** – Nejvyšší prioritní třída. Při spouštění podu s touto třídou je umožněno potlačit ostatní pody s nižší prioritou.
- **high-priority-np** – Nejvyšší prioritní třída, ale bez možnosti potlačování ostatních podů.
- **medium-priority-np** – Prostřední třída. Je výchozí pro všechny pody.
- **low-priority-np** – Nejnižší prioritní třída.

Prioritní třída se k instanci pipeline přiřazuje pomocí direktivy `pod` a její varianty `priorityClassName`. Příklad použití je na obrázku 4.13.

```
1  process priklad {
2
3    pod priorityClassName: 'low-priority-np'
4
5  }
```

Obrázek 4.13: Nastavení prioritní třídy. Zdroj: vlastní

4.6 Dokumentace REST API

REST API poskytuje synchronizovaná data buď uživatelskému rozhraní anebo externímu nástroji. Veškerá dat jsou odesílána i přijímána ve strukturovaném formátu JSON. Základní jednotkou je zdroj (*resource*), který může být obsažen v kolekci (*collection*). Každý zdroj i kolekce jsou identifikovatelné podle vlastního jednoznačného identifikátoru (URI) a lze s nimi manipulovat pomocí standardních HTTP metod pro CRUD operace. Názvy zdrojů a jejich URI jsou v množném čísle a anglickém jazyce.

Přehled dostupných koncových bodů REST API a možných metod je v následující tabulce 4.1.

Koncový bod	Metoda	Popis
`/api/workflows`	`GET`	Vrátí výpis všech instancí pipeline
`/api/workflows/0`	`POST`	Vytvoření instance pipeline
`/api/workflows/{id}`	`GET`	Vrátí instanci pipeline podle ID
`/api/workflows/{id}`	`POST`	Upravení instance pipeline podle ID
`/api/workflows/{id}`	`DELETE`	Smazání instance pipeline podle ID
`/api/workflows/{id}/upload`	`POST`	Nahrání vstupních souborů pro instanci pipeline
`/api/workflows/{id}/launch`	`POST`	Spuštění instance pipeline
`/api/workflows/{id}/resume`	`POST`	Opětovné spuštění instance pipeline
`/api/workflows/{id}/cancel`	`POST`	Zrušení běhu instance pipeline
`/api/workflows/{id}/log`	`GET`	Získání logu o průběhu instance pipeline
`/api/workflows/{id}/download`	`GET`	Stažení výstupních souborů (`.tar.gz`)
`/api/tasks`	`GET`	Výpis všech úloh
`/api/tasks`	`POST`	Uložení úlohy (využíváno Nextflow)

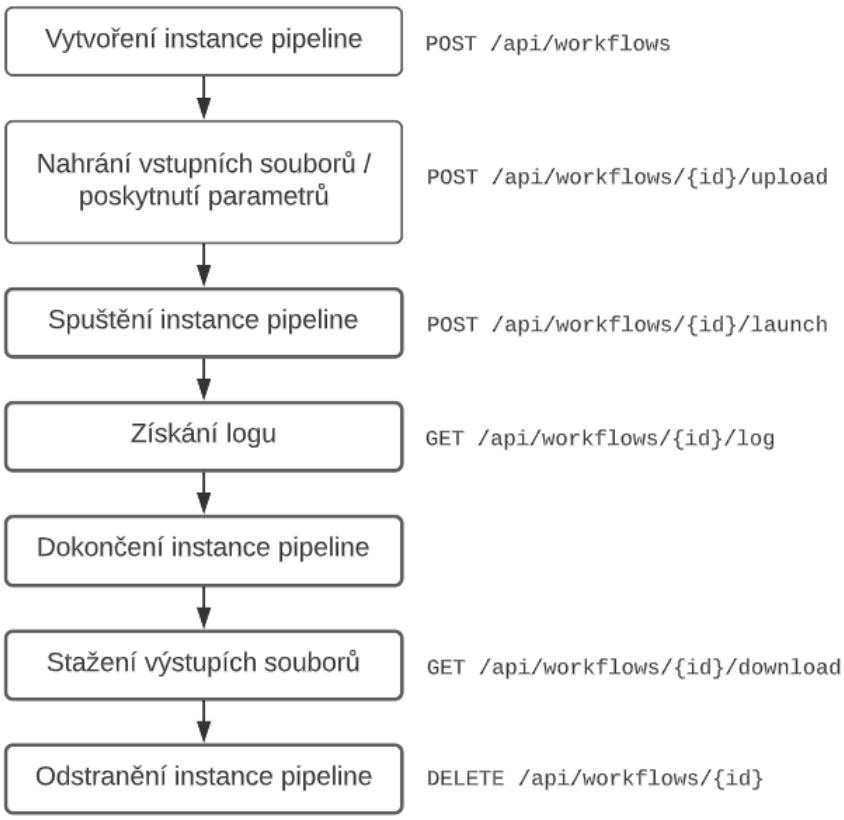
Tabulka 4.1: Přehled koncových bodů REST API. Zdroj: vlastní

4.7 Životní cyklus pipeline

Aby mohla být sekvenační data správně analyzována a pipeline úspěšně dokončena, je důležité dodržet korektní postup při práci s instancí pipeline. Proces zahrnuje následující kroky:

1. Jako první je potřeba zavolat koncový bod pro vytvoření instance pipeline a poskytnout potřebné informace. Povinný je odkaz (název podle tvaru zmíněném v kapitole 2.5.6) na pipeline, která se má pro instanci použít.
2. Poté je možné nahrát potřebné vstupní soubory (včetně `nextflow.config`) nebo přiřadit instanci pipeline parametry.
3. Následuje spuštění instance pipeline.
4. Jakmile je instance spuštěna, lze získat pomocí příslušného koncového bodu informace o průběhu pipeline (log). V ideálním případě mohou externí nástroje pro získání nejaktuálnějšího logu periodicky volat koncový bod.
5. Po dokončení má uživatel možnost stáhnout výstupní soubory vygenerované frameworkem Nextflow během exekuce pipeline. Nástroj soubory zkomprimuje do formátu *tarball* (`.tar.gz`).
6. Odstranění instance pipeline a přidružených dat je vhodným posledním krokem kvůli uvolnění paměti a zachování přehlednosti.

Obrázek 4.14 znázorňuje základní životní cyklus instance pipeline včetně koncových bodů a metod pro provedení jednotlivých kroků.



Obrázek 4.14: Základní životní cyklus instance pipeline. Zdroj: vlastní

4.8 Uživatelská příručka

Uživatelská příručka obsahuje přehled potřebných prerekvizit a postup jednoduchého nasazení nástroje. Nutno podotknout, že práce byla zpracovávána na platformě MacOS, a tudíž jsou pro tuto platformu i všechny uváděné návody. Postup se ale bude minimálně lišit od platformy Linux a je určitě v rámci platformy proveditelný.

Prerekvizity

- Git (`https://git-scm.com/download/mac`)
- Docker (`https://docs.docker.com/desktop/mac/install/`)
- Minikube (`https://minikube.sigs.k8s.io/docs/start/`)
- Kubectl (`https://kubernetes.io/docs/tasks/tools/install-kubectl-macos/`)
- Helm (`https://helm.sh/docs/intro/install/`)

Nasazení

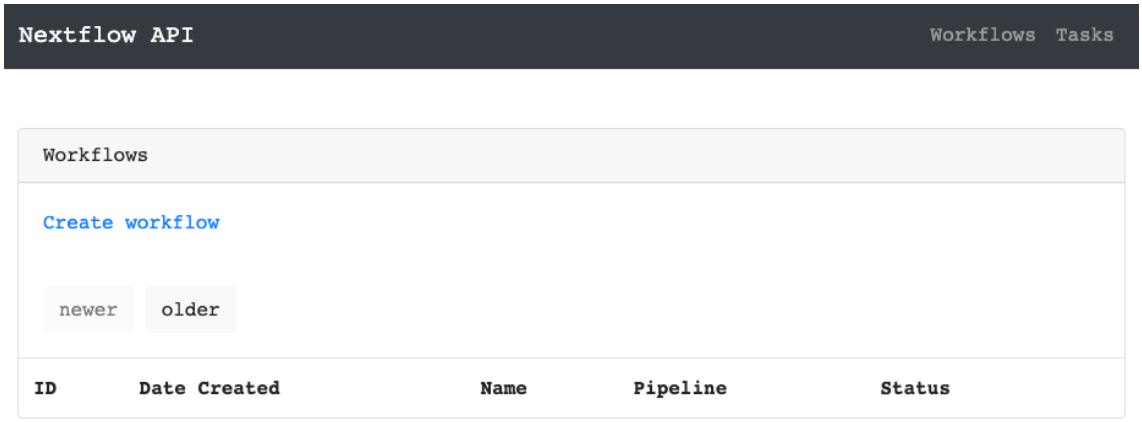
1. Stažení a nainstalování prerekvizit (oficiální odkazy na stažení v závorkách)
2. Stažení repozitáře projektu:
    - `git clone https://github.com/ihitch/DP.git`
3. Změnění pracovního adresáře na adresář projektu:
    - `cd DP`
4. Spuštění skriptu pro nasazení nástroje:
    - `./deploy.sh`
    - `./deploy.sh <nazev_obrazu>`

Tento krok má dvě varianty. První varianta skriptu použije k nasazení nástroje Docker obraz vytvořený v rámci této práce nahraný ve vzdáleném registru (Docker Hub). Skript spuštěný s argumentem `<nazev_obrazu>` vytvoří vlastní obraz pod poskytnutým jménem a nahraje ho do vzdáleného registru. Pro použití této varianty skriptu je ještě nutné změnit v souboru `DP/helm/values.yaml` proměnnou `Image` v nastavení služby `WebServer` na stejnou hodnotu jako při spuštění skriptu. Jakmile skript provede úspěšné nasazení, příkazová řádka vypíše adresu, pod kterou je nástroj dostupný.

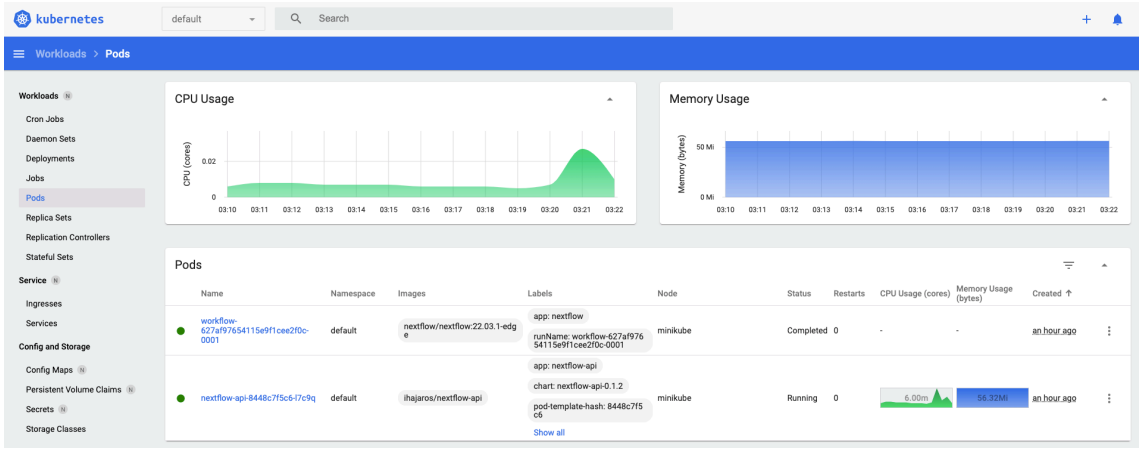
Detailnější popis nasazení je k dispozici v kapitole 4.5 Nasazení a pokud by uživatel měl zájem konfigurovat jakoukoliv ze služeb, lze si o tom podrobněji přečíst v kapitole 4.5.1 Konfigurace služeb.

5 Výsledky

Tato kapitola obsahuje dva scénáře, které demonstrují funkčnost a použití vytvořeného nástroje. Dále je zde zobrazena grafická podoba vzorového uživatelského rozhraní (obrázek 5.1) a služby Dashboard (obrázek 5.2) pro monitorování Kubernetes klastru.



Obrázek 5.1: Vzhled základní stránky uživatelského rozhraní. Zdroj: vlastní



Obrázek 5.2: Kubernetes Dashboard. Zdroj: vlastní

5.1 Scénáře použití

Scénáře jsou prováděny na již nasazeném nástroji a pro odesílání požadavků je použit nástroj pro příkazový řádek `curl` a nástroje `json_pp` a `jq` pro čitelnější formátování odpovědí.

5.1.1 Základní proces

Základní proces použití kopíruje životní cyklus pipeline zmíněný v části 4.7. Instance je vytvořena z pipeline `ihitch/pipeline-test`, jejíž kód je dostupný na odkazu: `https://github.com/ihitch/pipeline-test`.

1. Vytvoření instance pipeline (obrázek 5.3). Jediný povinný údaj je název referenční pipeline, je ale dobré si instance pojmenovávat pro lepší orientaci. Hodnotu `_id` vrácenou v odpovědi je dobré si uložit, pro další použití.
   - `curl -d '{"pipeline":"ihitch/pipeline-test", "name":"example"}' -H "Content-Type: application/json" -X POST http://127.0.0.1:49343/api/workflows/0`

```
→  ~ curl -d '{"pipeline":"ihitch/pipeline-test", "name":"example"}' -H "Content-Type: application/json"
 -X POST http://127.0.0.1:49343/api/workflows/0
{"_id": "627b24fa54115e9f1cee2f23"}%
```

Obrázek 5.3: Vytvoření instance. Zdroj: vlastní

2. Nahrání vstupních souborů (obrázek 5.4). V tomto případě jde o soubor `nextflow.config` a měl by vypadat přibližně jako na obrázku 5.5. Konfigurační soubor není nutné pokaždé nahrávat, je ho možné přidat do repozitáře s pipeline.
   - `curl -F upfile=@nextflow.config -X POST http://127.0.0.1:49343/api/workflows/627b24fa54115e9f1cee2f23/upload | json_pp`

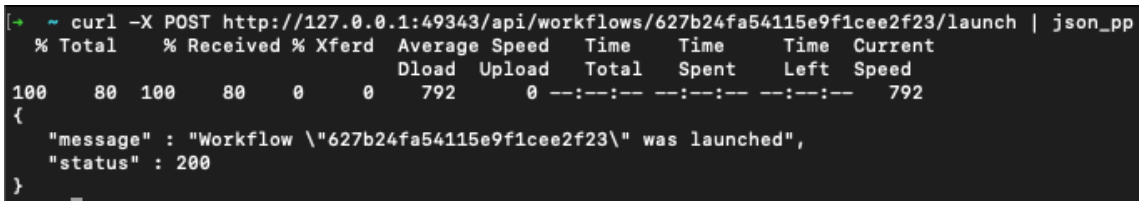
```
→  ~ curl -F upfile=@nextflow.config -X POST http://127.0.0.1:49343/api/workflows/627b24fa54115e9f1cee2f23/upload |
json_pp
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   477  100   126  100   351  10500  29250 --:--:-- --:--:-- --:--:-- 43363
{
   "status" : 200,
   "message" : "File \"['nextflow.config']\" was uploaded for workflow \"627b24fa54115e9f1cee2f23\" successfully"
}
```

Obrázek 5.4: Nahrání vstupních souborů. Zdroj: vlastní

```
1  manifest {
2    description = 'Nextflow example'
3    author = 'Jaroslav Iha'
4  }
5
6  process.container = 'nextflow/examples:latest'
7  process.cpus = 0.1
```

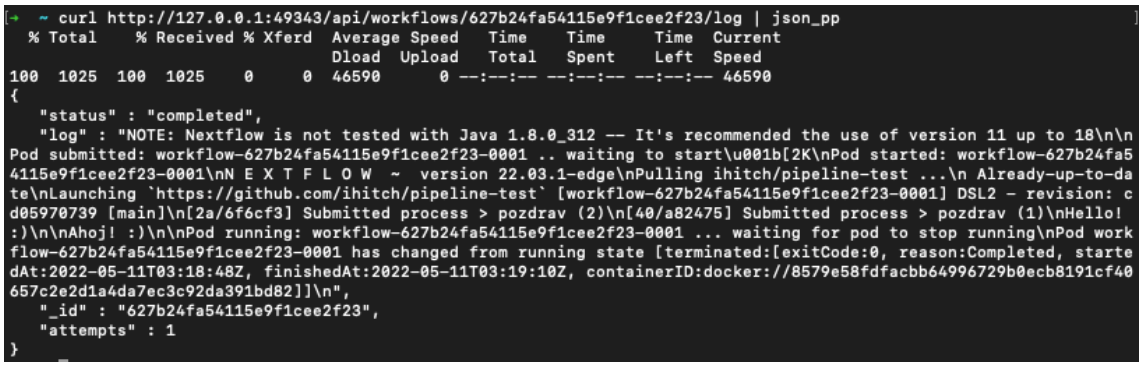
Obrázek 5.5: Soubor `nextflow.config`. Zdroj: vlastní

3. Spuštění instance pipeline (obrázek 5.6).
   - `curl -X POST http://127.0.0.1:49343/api/workflows/627b24fa54115e9f1cee2f23/launch | json_pp`



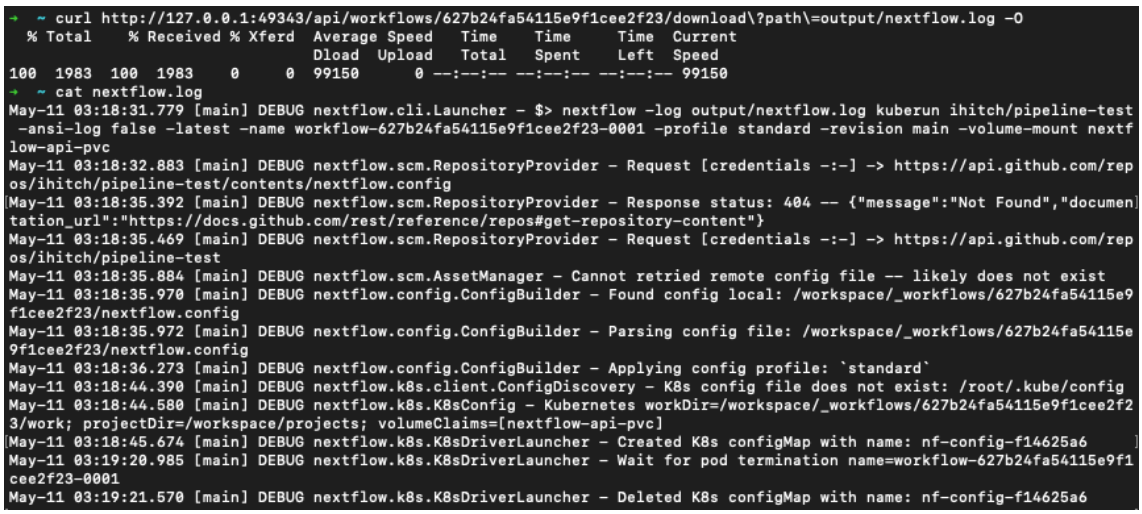
Obrázek 5.6: Spuštění instance pipeline. Zdroj: vlastní

4. Získání výpisu exekuce instance (obrázek 5.7).
   - `curl http://127.0.0.1:49343/api/workflows/627b24fa54115e9f1cee2f23/log | json_pp`



Obrázek 5.7: Získání logu. Zdroj: vlastní

5. Stažení výstupních souborů (obrázek 5.8). Možné soubory ke stažení lze zjistit pomocí požadavku pro výpis instance pipeline.
   - `curl http://127.0.0.1:49343/api/workflows/627b24fa54115e9f1cee2f23/download\?path\=output/nextflow.log -O`



Obrázek 5.8: Stažení logu. Zdroj: vlastní

6. Odstranění instance pipeline (obrázek 5.9).

- `curl -X DELETE http://127.0.0.1:49343/api/workflows/627b24fa54115e9f1cee2f23 | json_pp`

```
→  ~ curl -X DELETE http://127.0.0.1:49343/api/workflows/627b24fa54115e9f1cee2f23 | json_pp
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    79  100    79    0     0   4647      0 --:--:-- --:--:-- --:--:--  4647
{
   "status" : 200,
   "message" : "Workflow \"627b24fa54115e9f1cee2f23\" was deleted"
}
```

Obrázek 5.9: Odstranění instance. Zdroj: vlastní

5.1.2 Použití prioritních tříd

Tento příklad ukazuje použití prioritních tříd pro instance pipeline. Referenční pipeline je dostupná na adrese `https://github.com/ihitch/sleep-pipeline` a jde o jednoduchý skript, který spouští příkaz `sleep` na požadovaný počet sekund. Počet sekund, ale i další parametry, jako prioritní třída nebo požadovaná paměť se dají nastavit pro každou instanci samostatně, při jejím vytváření nebo upravování v poli `params_data`.

1. Byly vytvořeny 3 instance s takovými požadavky na výpočetní zdroje, aby mohla běžet vždy jen jedna a každá s jinou prioritní třídou (`low-priority-np`, `medium-priority-np` a `high-priority-np`). Výpis instancí byl pro lepší přehlednost zredukován pomocí nástroje `jq` jen na dvě pole (název, stav). Příkaz pro výpis:
   - `curl http://127.0.0.1:55955/api/workflows | jq ".[] | {name, status}" -`
2. Jako první byla spuštěna instance se střední prioritou, následně ta s nejnižší a jako poslední instance s prioritou nejvyšší.
3. Po dokončení první spuštěné instance (obrázek 5.10) Kubernetes automaticky naplánoval spuštění instance s nejvyšší prioritou, i když byl požadavek na její spuštění odeslán až jako poslední.

```
→  ~ curl http://127.0.0.1:55955/api/workflows | jq ".[] | {name, status}" -
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1044  100  1044    0     0  25463      0 --:--:-- --:--:-- --:--:-- 25463
{
  "name": "low-p",
  "status": "running"
}
{
  "name": "medium-p",
  "status": "completed"
}
{
  "name": "high-p",
  "status": "running"
}
```

Obrázek 5.10: Dokončení běhu první instance. Zdroj: vlastní

4. Jako druhý byl tedy dokončen běh instance s nejvyšší prioritou (obrázek 5.11).

```
→  ~ curl http://127.0.0.1:55955/api/workflows | jq ".[] | {name, status}" -
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1046  100  1046    0     0  30764      0 --:--:-- --:--:-- --:--:-- 30764
{
  "name": "low-p",
  "status": "running"
}
{
  "name": "medium-p",
  "status": "completed"
}
{
  "name": "high-p",
  "status": "completed"
}
```

Obrázek 5.11: Dokončení běhu druhé instance. Zdroj: vlastní

5. A instance s nejnižší prioritou byla dokončena (obrázek 5.12) jako poslední i přesto, že byla spouštěna jako druhá.

```
→  ~ curl http://127.0.0.1:55955/api/workflows | jq ".[] | {name, status}" -
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1046  100  1046    0     0  74714      0 --:--:-- --:--:-- --:--:-- 74714
{
  "name": "low-p",
  "status": "completed"
}
{
  "name": "medium-p",
  "status": "completed"
}
{
  "name": "high-p",
  "status": "completed"
}
```

Obrázek 5.12: Dokončení instance s nejnižší prioritou. Zdroj: vlastní

6 Diskuse

Vytvořený nástroj poskytuje rozhraní pro vytváření, spouštění, správu a monitorování instancí bioinformatických pipeline. Umožňuje automatickou správu výpočetních zdrojů, kterou má uživatel možnost také monitorovat. Nástroj je snadno nasaditelný a spustitelný v rámci několika jednoduchých kroků. Ačkoli je výsledné řešení funkční a splňuje předem definované požadavky, jedná se o první koncept, a rozhodně existuje prostor pro jeho zlepšování.

Celá architektura nástroje by se dala lepším způsobem optimalizovat. Aby se lépe držela mikroslužbové architektury, bylo by vhodné přesunout každou komponentu zvlášť do samostatného kontejneru. Umožnilo by to důkladnější řízení a správu výpočetních zdrojů a zlepšilo by to škálovatelnost celé aplikace.

Dalším krokem ke zlepšení bezpečnosti by mohlo být přidání autentizace a autorizace. Analyzovaná data jsou často té nejcitlivější povahy a pokud by se k nim mohl dostat kdokoliv, mohlo by to způsobit obrovské škody. Autorizace (přidělení uživatelských rolí) by mohla určovat, k jakým částem nástroje má uživatel přístup. Aby se nemohlo například stát, že by škodlivý uživatel vytvářel v Kubernetes klastru pody s nejvyšší možnou systémovou prioritou, a tím úplně znemožnil fungování nástroje.

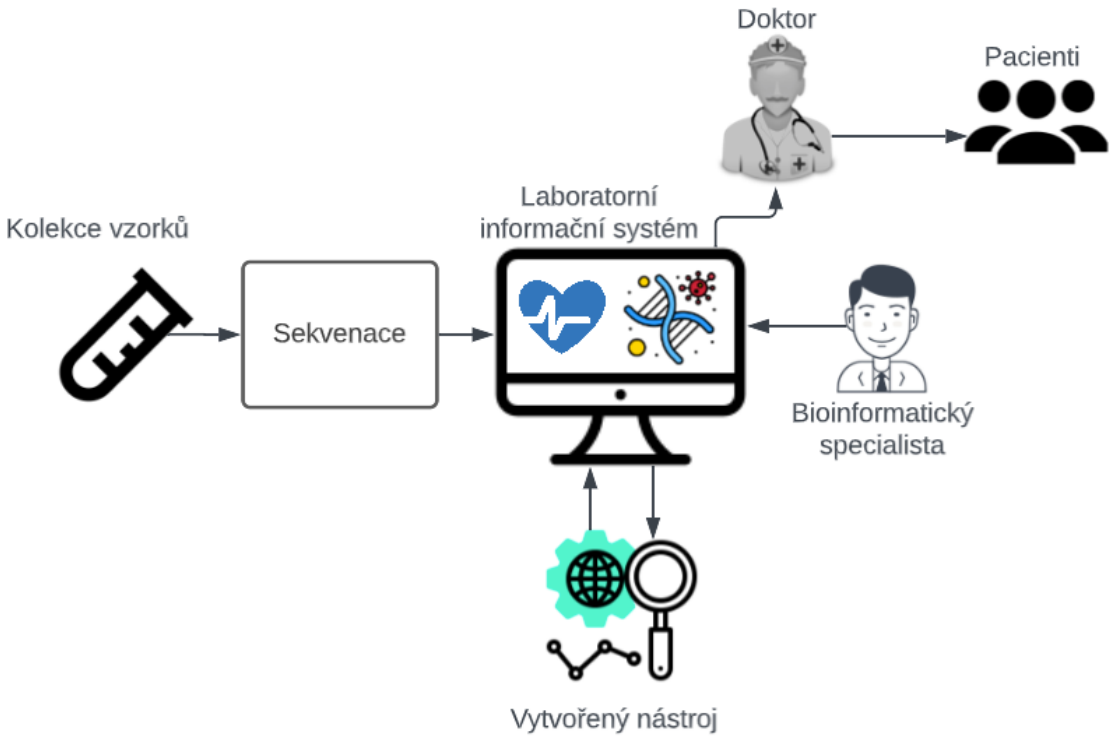
Momentálně je i lehkým omezením použití frameworku Nextflow. Nástroj ke správné funkčnosti používat verzi frameworku `22.03.1-edge`, což je nejnověji vydaná distribuce a není považována za stabilní. Mohou být například omezeny některé funkcionality, které by v případě stabilní verze fungovali bez problémů.

Přínos práce vidím v tom, že se mi nepodařilo najít jiný produkt stejného rázu, který by byl bezplatně dostupný. Pokud podobný nástroj existuje, tak jen v rámci komerčních platforem. Komerční řešení je ale mnohdy velice složité nebo dokonce nemožné integrovat do vlastních produktů, a tím ztrácí na použitelnosti. Využití služeb komerčních platforem navíc není úplně levnou záležitostí.

Pokud by byl k nasazení použit Kubernetes klastr od poskytovatele cloudových služeb (AWS, Google Cloud atd.), způsob, jakým nástroj funguje by umožnil korigování nákladů na provoz. Cloudové Kubernetes klastry se škálují automaticky s narůstajícími požadavky na výpočetní zdroje. Toto chování, ale rapidně zvyšuje neočekávané výdaje. Uživatel si může ke každé pipeline uložit parametry určující počet použitých procesorových jader a paměti. Společně s možností prioritizace instancí je toto cesta, jak mít náklady na provoz nástroje pod kontrolou.

Výhodou vytvořeného nástroje je vysoká integrovatelnost například do laboratorních informačních systémů nebo jiných nemocničních systémů. Bioinformatici nemusejí vyvíjet nové skripty pro analýzu dat, ale mohou jednoduše použít stávající. Stačí je jen

uzpůsobit pro spuštění pomocí frameworku Nextflow. Nasazení a uvedení nástroje do provozu je jednoduchým procesem a následná konfigurace je velice variabilní, což umožňuje vytvoření systému, který splňuje požadavky uživatele. Diagram, jak by integrace nástroje mohla vypadat, je znázorněn na obrázku 6.1.



Obrázek 6.1: Diagram integrace nástroje do laboratorního informačního systému. Zdroj: vlastní

7 Závěr

Jednotlivé cíle vytyčené v rámci diplomové práce byly úspěšně splněny. Hlavním záměrem bylo vytvořit univerzální nástroj, který by na základě standardizovaných postupů dokázal automatizovat bioinformatické procesy. V první části práce je popis a analýza aktuálně využívaných technologií a pravidel (kontejnerizace, REST atd.), na jejichž základě byla vytvořena výsledná podoba nástroje. Dalším záměrem bylo zajištění kontroly výpočetních zdrojů, což bylo dosaženo použitím technologií kontejnerů a orchestrace. Do architektury nástroje je začleněna platforma Kubernetes, která se stará o automatickou správu a řízení výpočetních prostředků jak pro fungování samotné aplikace, tak pro běh jednotlivých bioinformatických pipeline. K nástroji byla vytvořena dokumentace a uživatelská příručka s podpůrným skriptem pro jeho snadné nasazení a spuštění.

Seznam použité literatury

[1] GARIJO, Daniel, Sarah KINNINGS, Li XIE, Lei XIE, Yinliang ZHANG, Philip E. BOURNE, Yolanda GIL a Christos A. OUZOUNIS. Quantifying Reproducibility in Computational Biology: The Case of the Tuberculosis Drugome. PLoS ONE [online]. 2013, 8(11) [cit. 2022-04-30]. ISSN 1932-6203. Dostupné z: doi:10.1371/journal.pone.0080278

[2] DI TOMMASO, Paolo, Maria CHATZOU, Evan W FLODEN, Pablo Prieto BARJA, Emilio PALUMBO a Cedric NOTREDAME. Nextflow enables reproducible computational workflows. Nature Biotechnology [online]. 2017, 35(4), 316-319 [cit. 2022-04-30]. ISSN 1087-0156. Dostupné z: doi:10.1038/nbt.3820

[3] ROY, Somak, Christopher COLDREN, Arivarasan KARUNAMURTHY, et al. Standards and Guidelines for Validating Next-Generation Sequencing Bioinformatics Pipelines. The Journal of Molecular Diagnostics [online]. 2018, 20(1), 4-27 [cit. 2022-04-30]. ISSN 15251578. Dostupné z: doi:10.1016/j.jmoldx.2017.11.003

[4] FIELDING, Roy Thomas. Architectural styles and the design of network-based software architectures. Irvine, 2000. Order Number: AAI9980887. University of California.

[5] FREEMAN, Jonathan. What is an API? Application programming interfaces explained [online]. 2019 [cit. 2022-04-30]. Dostupné z: https://www.infoworld.com/article/3269878/what-is-an-api-application-programming-interfaces-explained.html

[6] WEBBER, Jim. REST in Practice. BABAR, Muhammad Ali a Ian GORTON, ed. Software Architecture [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, 2010, s. 7-7 [cit. 2022-04-30]. Lecture Notes in Computer Science. ISBN 978-3-642-15113-2. Dostupné z: doi:10.1007/978-3-642-15114-9_3

[7] FIELDING, Roy Thomas. REST APIs must be hypertext-driven [online]. 2008 [cit. 2022-04-30]. Dostupné z: https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

[8] HTTP request methods [online]. 2021-10-03 [cit. 2022-04-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

[9] HTTP response status codes [online]. 2022-02-18 [cit. 2022-04-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

[10] HTTP headers [online]. 2022-04-13 [cit. 2022-04-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

[11] CASALICCHIO, Emiliano a Stefano IANNUCCI. The state-of-the-art in container technologies: Application, orchestration and security. Concurrency and Computation: Practice and Experience [online]. 2020, 32(17) [cit. 2022-04-30]. ISSN 1532-0626. Dostupné z: doi:10.1002/cpe.5668

[12] IBM Cloud Education. Containerization [online]. 23 June 2021 [cit. 2022-04-30]. Dostupné z: https://www.ibm.com/cloud/learn/containerization

[13] PEARCE, Michael, Sherali ZEADALLY a Ray HUNT. Virtualization. ACM Computing Surveys [online]. 2013, 45(2), 1-39 [cit. 2022-04-30]. ISSN 0360-0300. Dostupné z: doi:10.1145/2431211.2431216

[14] MATELSKY, Jordan, Gregory KIAR, Erik JOHNSON, Corban RIVERA, Michael TOMA a William GRAY-RONCAL. Container-Based Clinical Solutions for Portable and Reproducible Image Analysis. Journal of Digital Imaging [online]. 2018, 31(3), 315-320 [cit. 2022-04-30]. ISSN 0897-1889. Dostupné z: doi:10.1007/s10278-018-0089-4

[15] DOUGLIS, Fred a Jason NIEH. Microservices and Containers. IEEE Internet Computing [online]. 2019, 23(6), 5-6 [cit. 2022-04-30]. ISSN 1089-7801. Dostupné z: doi:10.1109/MIC.2019.2955784

[16] LIU, Guozhi, Bi HUANG, Zhihong LIANG, Minmin QIN, Hua ZHOU a Zhang LI. Microservices: architecture, container, and challenges. In: 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C) [online]. IEEE, 2020, 2020, s. 629-635 [cit. 2022-04-30]. ISBN 978-1-7281-8915-4. Dostupné z: doi:10.1109/QRS-C51114.2020.00107

[17] DOCKER Inc. Docker overview [online]. [cit. 2022-04-30]. Dostupné z: https://docs.docker.com/get-started/overview/

[18] KURTZER, Gregory M., Vanessa SOCHAT, Michael W. BAUER a Attila GURSOY. Singularity: Scientific containers for mobility of compute. PLOS ONE [online]. 2017, 12(5) [cit. 2022-04-30]. ISSN 1932-6203. Dostupné z: doi:10.1371/journal.pone.0177459

[19] SYLABS Inc. Introduction to Singularity [online]. [cit. 2022-04-30]. Dostupné z: https://sylabs.io/guides/3.5/user-guide/introduction.html

[20] CASALICCHIO, Emiliano. Container Orchestration: A Survey. PULIAFITO, Antonio a Kishor S. TRIVEDI, ed. Systems Modeling: Methodologies and Tools [online]. Cham: Springer International Publishing, 2019, 2019-10-17, s. 221-235 [cit. 2022-04-30]. EAI/Springer Innovations in Communication and Computing. ISBN 978-3-319-92377-2. Dostupné z: doi:10.1007/978-3-319-92378-9_14

[21] ZHONG, Zhiheng a Rajkumar BUYYA. A Cost-Efficient Container Orchestration Strategy in Kubernetes-Based Cloud Computing Infrastructures with Heterogeneous Resources. ACM Transactions on Internet Technology [online]. 2020, 20(2), 1-24 [cit. 2022-04-30]. ISSN 1533-5399. Dostupné z: doi:10.1145/3378447

[22] IBM Cloud Education. Container Orchestration [online]. 27 May 2021 [cit. 2022-04-30]. Dostupné z: https://www.ibm.com/cloud/learn/container-orchestration

[23] The Kubernetes Authors. What is Kubernetes? [online]. 04 April 2022 [cit. 2022-04-30]. Dostupné z: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[24] The Kubernetes Authors. Kubernetes Components [online]. 30 April 2022 [cit. 2022-04-30]. Dostupné z: https://kubernetes.io/docs/concepts/overview/components/

[25] IBM Cloud Education. Kubernetes [online]. 22 July 2021 [cit. 2022-04-30]. Dostupné z: https://www.ibm.com/cloud/learn/kubernetes

[26] Seqera Labs. Basic concepts [online]. 2020-2022 [cit. 2022-04-30]. Dostupné z: https://www.nextflow.io/docs/latest/basic.html

[27] Seqera Labs. Processes [online]. 2020-2022 [cit. 2022-04-30]. Dostupné z: https://www.nextflow.io/docs/latest/process.html#process-page

[28] Seqera Labs. Channels [online]. 2020-2022 [cit. 2022-04-30]. Dostupné z: https://www.nextflow.io/docs/latest/channel.html

[29] Seqera Labs. Containers [online]. 2020-2022 [cit. 2022-04-30]. Dostupné z: https://www.nextflow.io/docs/latest/container.html

[30] Seqera Labs. Executors [online]. 2020-2022 [cit. 2022-04-30]. Dostupné z: https://www.nextflow.io/docs/latest/executor.html

[31] SchedMD. Documentation [online]. 14 December 2021 [cit. 2022-04-30]. Dostupné z: https://slurm.schedmd.com/documentation.html

[32] Seqera Labs. Pipeline sharing [online]. 2020-2022 [cit. 2022-04-30]. Dostupné z: https://www.nextflow.io/docs/latest/sharing.html

[33] Seqera Labs. Nextflow [online]. [cit. 2022-04-30]. Dostupné z: https://www.nextflow.io

[34] SHERMAN, Ben. Nextflow-API [online]. 10 July 2019 [cit. 2022-04-30]. Dostupné z: https://github.com/SciDAS/nextflow-api

[35] The Python Software Foundation. BeginnersGuide/Overview - Python Wiki [online]. 18 September 2019 [cit. 2022-04-30]. Dostupné z: https://wiki.python.org/moin/BeginnersGuide/Overview

[36] The Tornado Authors. Introduction [online]. 9 February 2019 [cit. 2022-04-30]. Dostupné z: https://www.tornadoweb.org/en/stable/guide/intro.html

[37] MongoDB, Inc. Introduction to MongoDB [online]. 2021 [cit. 2022-04-30]. Dostupné z: https://www.mongodb.com/docs/manual/introduction/

[38] The Kubernetes Authors. Deploy and Access the Kubernetes Dashboard [online]. 15 February 2022 [cit. 2022-04-30]. Dostupné z: https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

[39] The Kubernetes Authors. Welcome! | minikube [online]. 24 February 2022 [cit. 2022-04-30]. Dostupné z: https://minikube.sigs.k8s.io/docs/

[40] Helm Authors. Helm [online]. 2022 [cit. 2022-04-30]. Dostupné z: https://helm.sh

[41] GitHub, Inc. GitHub: Where the world builds software [online]. 2022 [cit. 2022-04-30]. Dostupné z: https://github.com

[42] Google LLC. AngularJS: Developer Guide: Introduction [online]. 2010-2020 [cit. 2022-04-30]. Dostupné z: https://docs.angularjs.org/guide/introduction

Seznam obrázků

Seznam tabulek