Uporaba zasebnega oblaka kot didaktično orodje za učinkovito raziskovalno delo in poučevanje

Gregor Cerar

Institut "Jožef Stefan", Jamova 39, Ljubljana, Slovenia
gregor.cerar@ijs.si

Private Cloud as a Didactic Tool for Effective Research and Teaching

Recent technological innovations have revolutionized education, enhancing teaching and learning experiences. Among the available tools, Jupyter Notebooks stand out as a powerful didactic tool for interactive computing, programming, and data analysis. However, their use becomes challenging on large projects due to hardware constraints. Public cloud services offer alternatives, but they come with significant limitations. Private cloud solutions like ours address cost and security concerns while providing flexibility. We use Kubernetes technology as the foundation of our private cloud solution, offering efficient application management, scalability, and resource flexibility.

1 Uvod

V preteklih letih je nenehen tehnološki napredek bistveno preoblikoval izobraževalni prostor ter izboljšal izkušnje pri poučevanju in učenju. Med mnogimi razpoložljivimi orodji so se Jupyter zvezki izkazali kot izjemno močno in vsestransko didaktično orodje za interaktivno računalništvo, programiranje in analizo podatkov [1, 2, 3].

Jupyter zvezek je odprt format dokumenta, ki temelji na JSON (*angl*., JavaScript Object Notation). Zvezki so strukturirani v zaporedje celic, kjer vsaka celica vključuje programsko kodo, opisno besedilo, enačbe ter obogatene izpise (*npr*.: besedilni izpisi, tabele, avdio, slike, animacije). Orodja, kot je JupyterLab, ponujajo platformo za interaktivno izvajanje kode, analizo podatkov in dokumentacijo v enem samem vmesniku, katerega končni izdelek je Jupyter zvezek. Zvezki podpirajo različne programske jezike (*npr*.: Python, R, Scala, C++) ter omogočajo uporabnikom iterativno pisanje in izvajanje celic kode (s pristopom REPL[1] ali WETL[2]), takojšen prikaz vmesnih rezultatov in s tem ustvarjanje pripovedno vodenih podatkovnih analiz, učnih gradiv in interaktivnih predstavitev. Zaradi te vsestranskosti in interaktivnosti so Jupyter zvezki močno didaktično orodje za učenje, izvajanje podatkovne znanosti in računalniških raziskav.

Zaradi teh izjemnih lastnosti smo se v našem raziskovalnem laboratoriju odločili vključiti Jupyter zvezke v naš izobraževalni ter raziskovalni proces. Na ta način želimo spodbujati študente in raziskovalce k dokumentiranju projektov ter omogočiti enostavno deljenje rezultatov raziskav.

Vendar pri velikih projektih, ko gre za obdelavo in analize velike količine podatkov na osebnem računalniku, postane uporaba zvezkov Jupyter precejšen izziv. Pogosto trčimo ob strojne omejitve, kot so pomanjkanje prostora za hrambo podatkov, delovnega spomina, omejena procesorska zmogljivost in dostop do računskih pospeševalnikov. Zaradi tega, delo postane oteženo ali celo nemogoče. Ker gre običajno za raziskave, analize ali prototipe v zgodnji fazi razvoja, rigorozne optimizacije na tej točki niso smiselne, saj te negativno vplivajo na hitrost razvoja eksperimenta. Kot rešitev se ponujata dve možnosti: izvajanje Jupyter zvezkov na grid, HPC (*angl*., high-performance computing) infrastrukturi ali v oblačni storitvi.

HPC infrastruktura, kot je v Sloveniji SLING, ponuja ogromno računske zmogljivosti. Ker je HPC velika investicija, se za povečanje izkoristka infrastrukture v HPC svetu uporabljajo rešitve čakalnih vrst kot je SLURM (*angl*., Simple Linux Utility for Resource Management). Računska naloga mora biti vnaprej zapakira v samozadosten paket metapodatkov, programske kode in vhodnih podatkov. Naloga v paketu se nato uvrsti na čakalni seznam. Takšen pristop ni združljiv s podatkovno usmerjenimi raziskavami (*angl*., data-driven research), ki stremijo k interaktivnemu programiranju in hitri povratni informaciji, zato Jupyter zvezkov ne moremo v celoti izkoristiti. Zaradi te omejitve so zvezki mnogo bolj razširjeni v storitvah v oblaku.

Javne oblačne storitve kot sta Google Colab[3] in Kaggle[4] so močno popularizirale rabo Jupyter zvezkov. Uporabnik lahko kadarkoli dostopa do storitve brez čakalnih vrst, ureja zvezke in uporablja računske zmogljivosti v oblaku, vse to preko spletnega brskalnika. Obe storitvi sta, v omejeni različici, prosto dostopna širši javnosti. Vendar zaradi navala uporabnikov se velikokrat zgodi, da storitev sama dodatno omejuje računske vire uporabnikom. S tem lahko postrežejo več uporabnikov, vendar pri tem vplivajo na kakovost storitve. Alternativa so podobne plačljive storitve po meri v javnem oblaku

[1] **REPL** read-eval-print loop
[2] **WETL** write-eval-think loop

[3] `https://colab.research.google.com/`
[4] `https://www.kaggle.com/`

(*npr.*: AWS, Azure, GCP, Alibaba Cloud), ki nudijo infrastrukturo po želji stranke. Storitve javnega oblaka imajo tudi nekaj slabosti. Visoke cene najemnin, kjer pomemben delež predstavlja hramba podatkov, težko predvidljivi stroški, na katere vpliva trg, in varnost zaradi morebitnega tveganja obdelave občutljivih podatkov.

Kot alternativa javnem oblaku, je zasebni oblak ključna rešitev za naslavljanje izzivov in pomislekov v zvezi s stroški in varnostjo. To je še posebej pomembno za raziskovalne laboratorije in podjetja, ki se soočajo z občutljivimi podatki ali zahtevajo visoko stopnjo prilagodljivosti. Hkrati organizacijam omogoča večjo preglednost in nadzor nad stroški, saj lahko samostojno oblikujejo svoje stroške na podlagi lastnih potreb in zmožnosti. Kljub zahtevi po tehničnem znanju in začetni naložbi v infrastrukturo, zasebni oblak dolgoročno ponuja izboljšano varnost, večji nadzor ter prilagodljivost, ki se odraža v bolj predvidljivih stroških.

Za vzpostavitev zasebnega oblaka je na izbiro vrsto tehnologij med katerimi so komercialne (*npr.*: VMware vSphere, Red Hat OpeShift, Cisco CloudCenter, IBM Cloud Private) in odprtokodne rešitve (*npr.*: OpenStack, Eucalyptus, Kubernetes). Med odprtokodnimi rešitvami je najbolj popularen Kubernetes.

Kubernetes[5] (krajše K8s) je odprtokodna platforma za avtomatizacijo, upravljanje in razporejanje aplikacij v vsebnikih (*angl.*, containers). S svojimi naprednimi funkcijami za orkestracijo omogoča učinkovito upravljanje aplikacij, njihovo avtomatsko širitev, nadzor nad njihovim delovanjem ter visoko razpoložljivost, kar olajšuje razvoj in vzdrževanje kompleksnih aplikacij v oblaku.

Za razliko od Docker[6] in Docker-compose[7], ki se osredotočata na izgradnjo, shranjevanje in zaganjanje posameznih vsebnikov, je Kubernetes veliko bolj celovita platforma za upravljanje vsebnikov z aplikacijam v obsežnejšem okolju, ki se razteza čez več računskih vozlišč. Medtem ko Docker ponuja enostavno ustvarjanje in delovanje posameznih kontejnerjev, ter Docker-compose omogoča definicijo več kontejnerjev kot aplikacijske enote, Kubernetes omogoča upravljanje celotnih sklopov teh aplikacijskih enot v različnih fazah življenjskega cikla. To vključuje samodejno razporejanje, dinamično prilagajanje glede na obremenitev, obnovitev v primeru napak in bolj napredno upravljanje storitev ter omrežja.

V našem raziskovalnem laboratoriju smo zaradi naraščajočih zahtev po večji računski zmogljivosti, ki prevladujejo v podatkovni znanosti, ter želje po ohranitvi prepoznavnega delovnega procesa, ki je prisoten pri Jupyter zvezkih, razvili lastno rešitev za zasebni računalniški oblak, ki temelji na Kubernetes tehnologiji.

V tem delu predstavljamo postavitev zasebnega oblaka z Jupyter zvezki, ki temelji na odprtokodnih rešitvah in po svoji uporabniški izkušnji spominja na že obstoječe plačljive oblačne storitve. Zasebni oblak mora izpolnjevati naslednje zahteve:

- **Razširljivost sistema:** Oblak mora omogočati enostavno dodajanje računskih vozlišč v gručo, ne da bi motili delujoč sistem. S tem storitvi omogočiti podporo večjim raziskovalnim projektom ali učnim skupinam.
- **Učinkovito upravljanje virov:** Sistem mora omogočati natančno dodeljevanje virov uporabnikom. Pri tem lahko administrator določi kompromis med ohlapno in strogo politiko dodeljevanja virov.
- **Izboljšana izkušnja sodelovanja:** Sistem mora omogočati preprosti deljenje Jupyter zvezkov med uporabniki in s tem spodbujati sodelovanje pri skupnih projektih in izmenjavo idej med raziskovalci in študenti.
- **Brez čakalnih vrst:** Sistem mora odpravljati potrebo po čakalni vrsti ter uporabnikom omogočati takojšen dostop do računskih virov po najboljših zmožnostih.

V preostanku članka razdelek 2 opiše našo arhitekturo zasebnega računskega oblaka grajeno z odprtokodnimi rešitvami. Razdelek 3 opiše arhitekturo iz vidika uporabnika, in razdelek 4 naredi povzetek in zaključek članka.

2 Arhitektura

Da zadovoljimo potrebam po razširljivosti sistema in učinkovitem upravljanju virov, predstavljenim v razdelku 1, smo se za osnovo našega zasebnega oblaka odločili za uporabo odprtokodne rešitve Kubernetes. Hkrati Kubernetes še dodatno izboljša funkcionalnost in dostopnost Jupyter zvezkov [4]. Za izboljšanje izkušnje sodelovanja, smo se odločili za uporabo odprtokodne rešitve JupyterHub, ki omogoča preprosto deljenje Jupyter zvezkov med uporabniki in tako spodbuja sodelovanje pri skupnih projektih in idejah. Kubernetes omogoča učinkovito in skalabilno upravljanje JupyterHub okolja za raziskave in poučevanje. V tem razdelku bomo podrobneje predstavili arhitekturo sistema, ki združuje/povezuje Jupyter zvezke, JupyterHub in Kubernetes, ter podrobneje predstavili odločitve pri zasnovi arhitekture. V nadaljevanju tega razdelka predstavljamo posamezne storitve znotraj naše infrastrukture.

Preglednica 1 v prvem stolpcu podaja vse storitve, ki smo jih morali zagotoviti za delovanje sistema. V drugem stolpcu so navedene odprtokodne rešitve, ki te storitve lahko zagotavljajo. V debeljenem tisku so označene izbrane storitve, ki so bile uporabljene v našem zasebnem oblaku. Rešitve smo izbirali glede na določene kriterije. Najprej smo opravili pregled tehnologij in rešitev, ki so se že uporabljale v sorodnih projektih. Pri izboru smo se omejili na odprtokodne rešitve, ki so bile preizkušene v zasebnih oblakih na lastni infrastrukturi. Pomemben dejavnik pri odločanju je bil tudi vpogled v popularnost projektov, ki smo jo merili s številom zvezdic v repozitorijih, številom razcepitev (*angl.*, forks) projekta ter stopnjo aktivnosti razvoja na GitHub-u. Pri odločitvi nismo sledili enotni empirični metriki, temveč smo upoštevali več dejavnikov, da smo zagotovili celovito presojo rešitev.

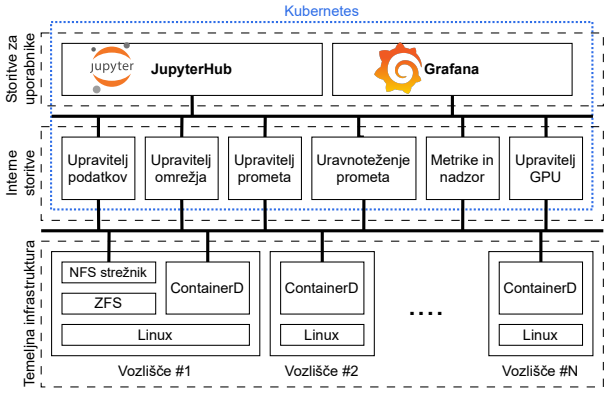
Shema na sliki 1 prikazuje visokonivojsko predstavitev našega zasebnega oblaka in infrastrukturo v treh ni-

[5] https://kubernetes.io/
[6] https://www.docker.com/
[7] https://docs.docker.com/compose/

vojih. Na prvem nivoju so heterogena računska vozlišča, ki so del *t.i.* temeljne infrastrukture. Posamezno vozlišče ima svoj operacijski sistem in na njem teče del platforme Kubernetes. Drugi nivo zajema interne Kubernetes storitve, ki so potrebne za delovanje in uporabnik nima nikoli neposrednega stika z njimi. Zadnji, tretji nivo, pa obsega storitve, ki jih koristijo končni uporabniki.

Tabela 1: Storitve in uporabljene tehnologije.

Storitev	Rešitve (**Uporabljeno**)
Rešitev na ključ?	**Po meri**, NVIDIA DeepOps
Temeljna infrastruktura	
Operacijski sistem	**Ubuntu**, RHEL, NixOS, Talos
Hramba podatkov	**ZFS**, GlusterFS, Lustre, CEPH, iSCSI
Upravljanje sistemov	**Ansible**, Terraform, Puppet, Chef
Interne storitve	
K8s distribucija	**"vanilla"**, MicroK8s, OpenShift, Rancher
K8s namestitev	**Helm**, Kustomize
Upravitelj omrežja	**Calico**, Canal, Flannel, Weave
Upravitelj podatkov	**csi-driver-nfs**, Rook, OpenEBS
Uravnoteženje prometa	**MetalLB**, cloud provider specific
Upravitelj prometa	**Nginx**, Traefik
Upravitelj GPU	**NVIDIA GPU-Operator**
Storitve za uporabnike	
Upravitelj JupyterHub	**Z2JH (Zero to JupyterHub)**
Metrike in nadzor	**kube-prometheus-stack**, InfluxDB



Slika 1: Logična shema infrastrukture v treh nivojih. Spodaj je temeljna infrastruktura, interne Kubernetes storitve in na vrhu storitve izpostavljene uporabnikom.

2.1 Rešitev na ključ – DA ali NE?

Pri načrtovanju zasebnega oblaka smo najprej preučevali rešitve na ključ (*angl.*, turnkey solution), med katerimi je bila tudi NVIDIA DeepOps[8]. Kljub njegovim prednostim smo se odločili za izgradnjo lastne prilagojene rešitve iz naslednjih razlogov. DeepOps sicer predstavlja odlično rešitev na ključ z dobro vzdrževano izvorno kodo na GitHubu in ponuja tudi komercialno podporo. Vendar začetna nastavitev zahteva prilagoditev konfiguracijskih datotek, vključno z uporabo priloženih Ansible skript za avtomatizirano preoblikovanje Linux distribucije in namestitev dodatnih potrebnih programskih paketov.

Pomanjkljivosti rešitve DeepOps so njena kompleksnost in zakritost podrobnosti. Gre za izjemno zapleteno rešitev, ki se trudi biti vsestranska in "enostavna" za uporabo, vendar to vodi v skrivanje funkcionalnosti in pomanjkanje dokumentacije o notranjih orodjih. To lahko povzroči težave pri odpravljanju težav ali nadgradnji, kar zahteva ročno posredovanje, saj je za obvladovanje sistema potrebno temeljito razumevanje Linuxa in Kubernetesa. Zaradi tega smo se odločili za začetek z minimalistično rešitvijo, s časom pa nameravamo sistem razširiti, da bomo bolje razumeli delovanje infrastrukture.

2.2 Temeljna infrastruktura

Temeljna infrastruktura našega zasebnega oblaka je zasnovana na več ključnih komponentah, ki omogočajo učinkovito in skalabilno delovanje sistema. V tem razdelku bomo podrobneje predstavili osnovne gradnike, vključno z izbiro orodij za upravljanje vsebnikov in delitev virov, ki so ključne za za delovanje platform Kubernetes.

Operacijski sistem: Za naš sistem smo se odločili za Ubuntu Server, ki temelji na Linux distribuciji Debian. Prednost široke uporabe Debian temelječih Linux distribucij je v obilju dostopnih virov znanja in podpore, kar olajša reševanje morebitnih težav. Med alternativami, kot je NixOS, ki omogoča ponovljive sisteme na osnovi opisov in receptov (*angl.*, binary reproducible system), ter različnimi RHEL distribucijami, smo pretehtali tudi distribucijo Talos, specializirano za Kubernetes. Vendar smo zaradi mladosti projekta in posledičnih tveganj raje ostali pri odločitvi za Ubuntu Server.

Upravljanje vsebnikov: Za upravljanje vsebnikov smo izbrali *ContainerD*, ki se uporablja tudi v DeepOps rešitvi in je uradno podprt s strani NVIDIA-e. Gre za odprtokodno orodje, ki implementira CRI (*angl.*, Container Runtime Interface) vmesnik za komunikacijo med operacijskim sistemom in Kubernetes za učinkovito in zanesljivo upravljanje vsebnikov.

Hramba podatkov: Za hrambo podatkov smo izbrali ZFS (*angl.*, Zettabyte File System), ki je postal del enega od vozlišč. Čeprav se v HPC svetu v večini uporabljajo HDFS, Gluster, Luster ali Ceph, ti ne podpirajo vseh naprednih lastnosti datotečnega sistema, kot jih ponuja ZFS. Ta vključuje kontrolne točke (*angl.*, checkpoints), deduplikacijo podatkov, stiskanje, COW (*angl.*, copy-on-write) sistem za preprečevanje izgube podatkov pri pisanju, imunost na obračanje bitov (*angl.*, silent bit-rot), možnost uporabe diskov kot redundance za primere mehanskih okvar in uporaba hitrih SSD naprav kot predpomnilnika. Prav tako omogoča enostavno ročno posredovanje v primeru incidentov. Vendar pa ZFS ne omogoča raztezanja čez več vozlišč, kar predstavlja tveganje za izpad gruče v primeru okvare vozlišča, ki hrani podatke (*angl.*, single point of failure). Za dostop do hrambe podatkov iz Kubernetes-a smo uporabili NFS

[8] https://github.com/NVIDIA/deepops

(*angl*., Network File System) strežnik, ki je del Linux jedra. NFS smo izbrali tudi zato, ker je eden redkih načinov, ki omogoča vezavo več vsebnikov na isto pritrdilno točko (*angl*., mounting point).

Upravljanje sistemov: Za oddaljeno upravljanje in konfiguracijo vozlišč uporabljamo Ansible[9]. Izbrali smo ga zaradi njegove razširjenosti v drugih pomembnih odprtokodnih projektih, kot tudi zaradi pozitivnih izkušenj iz preteklih projektov.

2.3 Notranje storitve

V Kubernetesu so notranje (interne) storitve tiste, ki niso namenjene končnim uporabnikom, temveč so ključnega pomena za delovanje samega sistema. Te storitve delujejo v ozadju in skrbijo za pomembne funkcionalnosti, ki omogočajo stabilno delovanje ter upravljanje okolja vsebnikov. V tem podrazdelku bomo predstavili ključne storitve v okviru Kubernetesa in razložili njihovo vlogo v naši infrastrukturi. Za vsako storitev bomo opisali njeno glavno funkcionalnost ter preučili alternative, ki smo jih raziskali pri naši odločitvi.

Kubernetes distribucija: Pri izbiri Kubernetes distribucije smo preučevali tri možnosti: Canonical MicroK8s, Red Hat OpenShift in osnovno "vanilla" distribucijo Kubernetesa. "Vanilla" Kubernetes predstavlja nespremenjeno različico, ki je na voljo neposredno v Googlovem repozitoriju in ne vključuje predhodno nameščenih aplikacij ali vtičnikov. To omogoča izbiro tistih, ki najbolje ustrezajo našim potrebam.

MicroK8s je odlična rešitev za hitro eksperimentiranje in vzpostavitev sistema na manjših napravah z omejenimi viri. Vendar pa ima veliko že vnaprej nameščenih aplikacij ter uporablja Canonicalov paketni sistem Snap, kar lahko otežuje prilagajanje nastavitvenih datotek in dostop do zunanjih storitev, kot je bil v našem primeru NFS strežnik.

OpenShift smo izločili zaradi kompleksnosti upravljanja varnostnih profilov, ki so za naš primer uporabe pretirani, s čimer bi morali vložiti precej truda v implementacijo teh profilov za vsako storitev. Zato smo se odločili za osnovno "vanilla" distribucijo Kubernetesa, ki nam omogoča bolj prilagodljivo in enostavno prilagajanje sistema našim specifičnim potrebam.

Kubernetes nameščanje paketov: Za opis implementacije storitev in vtičnikov v Kubernetesu uporabljamo YAML konfiguracijske datoteke, ki jih nato preko ukazne vrstice posredujemo Kubernetesu. Nekatere storitve so lahko precej zapletene, zato so razvijalci ustvarili pakete storitev, ki jih lahko prilagodimo s pomočjo parametrov. Najbolj razširjen paketni sistem je Helm, ki omogoča bolj prenosljive in prilagodljive pakete storitev. Helm uporablja YAML datoteke kot predloge (*angl*., templates), ki jih nato izpolni glede na podane parametre ter jih posreduje Kubernetesu.

Upravitelj omrežja (*angl*., network operator): Za upravljanje zankastega (*angl*., mesh) omrežja znotraj sistema Kubernetes, ki med seboj povezuje storitve, smo se odločili za uporabo odprtokodnega orodja Calico. To se je izkazalo za najbolj primerno rešitev glede na njegovo razširjenost in funkcionalnosti. Med sorodnimi projekti se pogosto uporabljata Calico ali Flannel. Flannel je minimalističen in deluje kot omrežno stikalo (*angl*., network switch) s pomočjo tehnologij, kot sta *open vSwitch* ali *VXLAN*. Nasprotno pa Calico usmerja promet kot omrežni usmerjevalnik (*angl*., network router). Zlasti v primeru infrastrukture na različnih lokacijah ali hibridnih oblačnih storitev se Calico izkaže kot boljša izbira.

Upravitelj shrambe (*angl*., storage operator): Za učinkovito upravljanje shrambe v okviru sistema Kubernetes smo se odločili za uporabo *csi-driver-nfs*, ki omogoča izrabo NFS strežnika, nameščenega na enem od vozlišč. S to rešitvijo zagotavljamo neprekinjen dostop do trajne shrambe za vsako storitev, ki jo potrebuje znotraj našega oblaka.

Izbira *csi-driver-nfs* se je izkazala za najbolj primerno, saj smo že imeli vzpostavljen NFS strežnik na enem od vozlišč. To nam je omogočilo preprosto in centralizirano upravljanje shrambe za vse storitve v okviru Kubernetes. S centralizacijo pridemo do številnih prednosti in hkrati tudi izzivov. Med slednjimi velja omeniti ranljivost sistema ob morebitnem izpadu vozlišča, ki hrani podatke. Kljub temu pa centralizacija omogoča lažje reševanje morebitnih težav in izvajanje varnostnih kopij.

Uravnoteženje vhodnega prometa (*angl*., ingress loadbalancer): Za zagotovitev uravnoteženega vhodnega prometa med vstopnimi točkami v naši Kubernetes gruči smo se odločili za uporabo rešitve MetalLB. Po temeljitem iskanju nismo našli smiselne alternative. Večina dostopne dokumentacije na spletu se osredotoča na postavljanje infrastrukture na javnih oblakih, kot sta AWS in Azure, ter uporabo rešitev, ki so prilagojene zahtevam ponudnikov javnih oblakov. Ker pa temelji naša infrastruktura na lastni strojni opremi (*t.i.* "bare-metal"), smo se odločili za MetalLB, ki se je izkazal kot zanesljivo in učinkovito orodje za usmerjanje prometa med našimi vstopnimi točkami v Kubernetes gručo.

Upravitelj (vhodnega) prometa (*angl*., ingress operator): Kubernetes omogoča komunikacijo med storitvami preko zasebnih IPv4 ali IPv6 omrežnih naslovov. Zaradi varnostnih razlogov neposreden dostop do teh naslovov ni dovoljen, zato se promet, podobno kot pri NAT (*angl*., network address translation), preusmeri preko upravitelja vhodnega prometa. Dostop do storitev je možen preko domenskih imen, ki jih upravitelj razreši in usmeri promet k želeni storitvi. Med najpogostejšimi rešitvami za upravljanje vhodnega prometa so NGINX in Traefik. V našem primeru smo izbrali NGINX, vendar je vmesnik upraviteljev standardiziran, zato skoraj ni razlik med rešitvami. Ne glede na izbrano rešitev administrator v manifestu uporabniške storitve določi novo poddomeno, nato pa upravitelj avtomatično preusmeri promet na ustrezne IP naslove.

Upravitelj GPU (*angl*., GPU-operator): Za učinkovito

[9] `https://www.ansible.com/`

upravljanje dostopa do računskih pospeševalnikov smo se odločili za uporabo uradnega kompleta storitev NVIDIA GPU-Operator. Ta komplet storitev omogoča dve možnosti namestitve NVIDIA gonilnikov. Prva možnost je uporaba gostiteljskih gonilnikov, medtem ko druga vključuje gonilnike, ki so zapakirani v vsebnike. Sprva smo se odločili za prvo možnost, saj smo želeli omogočiti tudi uporabo pospeševalnikov zunaj okvira Kubernetesa. Vendar smo se zaradi težav s konfliktnimi verzijami gonilnikov odločili, da bomo uporabili gonilnike, ki jih ponuja GPU-Operator.

2.4 Storitve za uporabnike

V tem razdelku predstavljamo izbrane storitve, ki so na voljo uporabnikom našega zasebnega oblaka in omogočajo učinkovito izvajanje in upravljanje svojih raziskovalnih in poučevalnih projektov.

JupyterHub predstavlja eno izmed ključnih storitev v našem zasebnem oblaku, ki omogoča uporabnikom enostaven dostop do računskih virov, podatkov in Jupyter zvezkov za potrebe raziskovanja in poučevanja. Za uresničitev JupyterHub-a uporabljamo implementacijo Z2JH (Zero-to-JupyterHub), ki jo razvija ekipa raziskovalcev na Univerzi v Berkeleyju v sodelovanju z Jupyter skupnostjo. Ta rešitev omogoča hitro vzpostavitev in vzdrževanje.

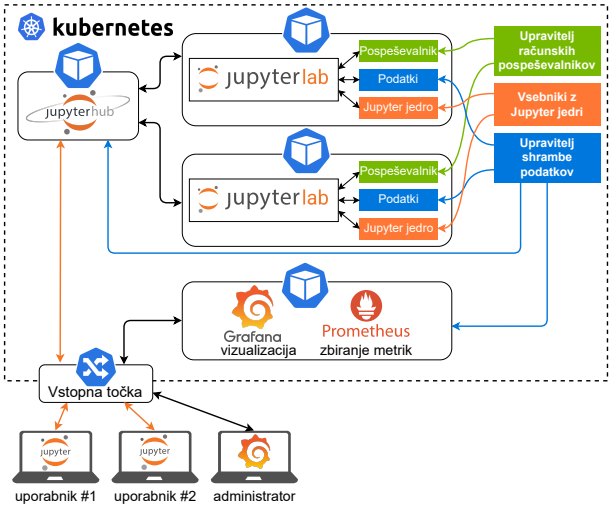
Vsak posamezen uporabnik dobi dostop do izolirane instance vsebnika, do katere dostopa preko svojega uporabniškega imena in gesla. Ta izolirana instanca ponuja okrnjeno Linux okolje ter omejen dostop do interneta. Kubernetes nato skrbi za zagotavljanje dostopa do deljenih podatkovnih virov, skupnih map ter uporabo računskih pospeševalnikov.

Uporabniški vmesnik JupyterHub-a je podoben storitvam kot sta Google Colab ali Kaggle. Ob vstopu v izolirano instanco je JupyterLab že zagnan, uporabnik pa ima prav tako dostop do Linux terminala. Dodatna orodja in programske pakete je mogoče namestiti preko ukazov pip, Conda ali Mamba.

Grafana je ključna storitev v našem zasebnem oblaku, ki omogoča enostaven prikaz trenutne obremenitve računske gruče ter razpoložljivosti računskih pospeševalnikov. Ta platforma za vizualizacijo podatkov omogoča uporabnikom pregledno in transparentno prikazovanje informacij, kar jim olajša sprejemanje odločitev glede uporabe virov in optimizacije njihovega dela. Uporaba Grafane zagotavlja učinkovito in jasno nadzorovanje virov, kar prispeva k izboljšani uporabniški izkušnji in učinkovitemu delovanju celotnega sistema.

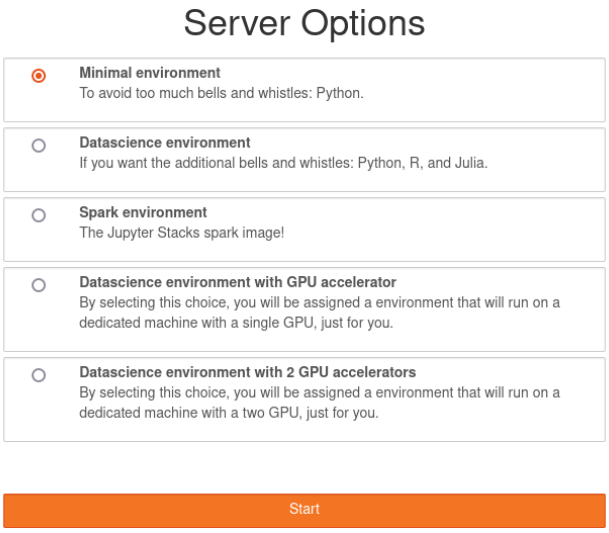
3 Pogled s strani uporabnika storitev

Pri uporabi naše rešitve zasebnega računskega oblaka, uporabniki doživljajo preprosto in intuitivno izkušnjo. Kot je predstavljeno na sliki 2. Za dostop do storitev, uporabnik zažene svoj spletni brskalnik in vpiše *hub.{domena}* za dostop do JupyterHub ali *monitor.{domena}* za Grafano. Kubernetes pa zahtevo uporabnika usmeri do pravih storitev.



Slika 2: Predstavljena oblačna storitev s perspektive uporabnika. Preko skupne vstopne točke uporabniki dostopajo do željenih storitev. Uporabnika #1 in #2 svojih instanc JupyterLab preko JupyterHub. Administrator dostopa do Grafane za vpogled v stanje infrastrukture.

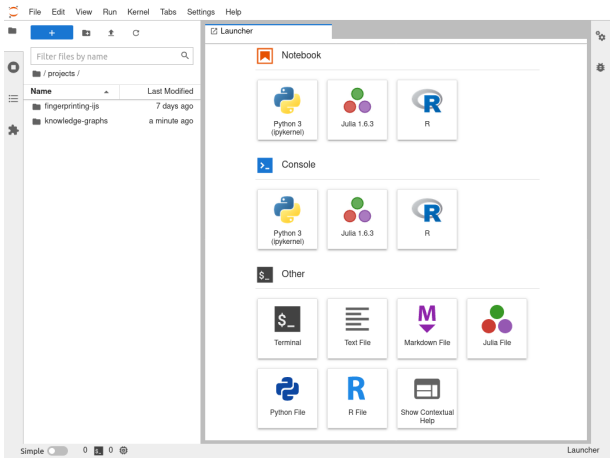
Ko se uporabnik prijavi v JupyterHub, mu ta najprej ponudi nabor profilov med katerimi lahko izbira (slika 3). Ti profili nudijo pakete prednameščenih orodij in nabor računskih virov, vključno s procesorsko močjo, pomnilnikom in računskimi pospeševalniki.



Slika 3: V storitvi JupyterHub uporabnik izbere profil okolja za delo, sistem pa poskrbi za dostop uporabnikovih podatkov in na željo uredi dostop do nedeljenega računskega pospeševalnika.

Uporabnik lahko izbere želeni profil, ki najbolje ustreza njihovim raziskovalnim ali izobraževalnim zahtevam. Na voljo so različne možnosti, ki segajo od osnovnih profilov za manj zahtevne naloge do zmogljivejših profilov za zahtevnejše računske operacije.

Po izbiri profila uporabnik vstopi v svoj delovni prostor, kjer ima na voljo širok nabor razpoložljivih jeder (*angl.* kernel) za izvajanje svojih Jupyter zvezkov (slika 4). Raznolik nabor jeder omogoča uporabnikom



Slika 4: Po izbiri profila okolja za delo ima uporabnik dostop do različnih Jupyter jeder (*npr.*: Python, Julia, R), svojih podatkov in zvezkov.

uporabo tistega programskega jezika v katerem se čutijo bolj učinkovite pri raziskovanju ali učenju. Vsak uporabnik se nahaja v izoliranjem okolju, ki zagotavlja, da se uporabniki ne motijo med seboj za najboljšo uporabniško in didaktično izkušnjo.

Glede na trenutno konfiguracijo uporabniki nimajo omejitev pri uporabi procesorskih enot in pomnilnika, kar jim omogoča nemoten izvajanje jupyter zvezkov. Dostop do računskih pospeševalnikov je urejen preko sistem rezervacije, zaradi omejenega število pospeševalnikov.

Za transparenten vpogled v razpoložljivost infrastrukture ima uporabnik dostop samo za branje (*angl.* read-only) do nadzorne plošče Grafana (slika 5), kjer lahko opazuje večino metrik infrastrukture.

Slika 5: Nadzorna plošča Grafane s pregledom zasedenosti virov v zasebnem računskem oblaku.

4 Zaključki

V tem članku smo predstavili našo inovativno rešitev zasebnega oblaka, ki temelji na tehnologiji Kubernetes. Naša rešitev prinaša učinkovito in skalabilno delovno okolje za uporabo Jupyter zvezkov, ki služijo kot močno didaktično orodje za pripovedno vodene podatkovne analize, izdelavo učnih gradiv ter interaktivne predstavitve. Poleg tega naš sistem omogoča hkratno souporabo računskih virov, kar bistveno poveča izkoriščenost celotne infrastrukture.

Storitev JupyterHub na platformi Kubernetes omogoča ne le enostaven dostop do delovnega okolja, ampak tudi izolacijo med uporabniki, kar omogoča nemoteno delo in raziskovanje. Uporabniki lahko koristijo shranjevalni prostor ter skupne mape za deljenje datotek, kar spodbuja sodelovanje in skupinsko delo. Dodatno imajo dostop do računskih pospeševalnikov, kadar ti niso zasedeni.

V članku smo predstavili ključne komponente, arhitekturo ter odločitve pri načrtovanju, hkrati pa smo razkrili izbor tehnologij, ki so vodile k učinkovitemu delovanju naše oblačne rešitve in izjemni uporabniški izkušnji. Pri izbiri tehnologij smo se osredotočili na odprtokodne in preizkušene platforme, ki so se izkazale za zanesljive in učinkovite v našem okolju. Kubernetes kot osrednja platforma omogoča skalabilno upravljanje vsebnikov ter visoko razpoložljivost. JupyterHub pa zagotavlja enostaven dostop do storitev ter olajšuje upravljanje uporabnikov.

Za prihodnost načrtujemo nadgradnjo naše rešitve z dodatnimi storitvami in tehnologijami, ki bodo še izboljšale uporabniško izkušnjo ter povečale zmogljivost našega oblaka. Hkrati bomo ostali odprti za nove tehnologije in pristope, ki bodo prispevali k še boljšemu delovanju naše zasebne oblačne rešitve za raziskave in poučevanje. Meritve iz storitve Prometheus bodo ključne za analizo izkoriščenosti infrastrukture in razumevanje, v kolikšnem deležu uporabniki tekmujejo za računske vire.

Zahvale

Delo sta financirala Javna agencija za znanstvenoraziskovalno in inovacijsko dejavnost Republike Slovenije (ARIS) v okviru P2-0016, ter Evropska komisija s projektom številka 101096456 (NANCY).

Literatura

[1] J. M. Perkel, "Why Jupyter is data scientists' computational notebook of choice," *Nature*, vol. 563, no. 7732, pp. 145–147, 2018.

[2] K. M. Mendez *et al.*, "Toward collaborative open data science in metabolomics using Jupyter Notebooks and cloud computing," *Metabolomics*, vol. 15, no. 10, pp. 1–16, 2019.

[3] B. E. Granger *et al.*, "Jupyter: Thinking and Storytelling With Code and Data," *Computing in Science & Engineering*, vol. 23, no. 2, pp. 7–14, 2021.

[4] M. Bussonnier, "Jupyter and HPC: Current state and future roadmap," Mar 2018. [Online]. Available: https://www.exascaleproject.org/event/jupyter/