

Sem vložte zadání Vaší práce.





**FAKULTA  
INFORMAČNÍCH  
TECHNologiÍ  
ČVUT V PRAZE**

Bakalářská práce

## **Kubernetes klastr pro lámání hesel**

*Tomáš Klas*

Katedra informační bezpečnosti (KIB)

Vedoucí práce: Ing. Jiří Buček, Ph.D.

13. března 2020



---

## Poděkování

Doplňte, máte-li komu a za co děkovat. V opačném případě úplně odstraňte tento příkaz.



---

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. března 2020

.....

České vysoké učení technické v Praze

Fakulta informačních technologií

© 2020 Tomáš Klas. Všechna práva vyhrazena.

*Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.*

### **Odkaz na tuto práci**

Klas, Tomáš. *Kubernetes klastr pro lámání hesel*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.



---

## Abstrakt

Hlavní náplní práce je nakonfigurování klastru pro lámání hesel. Tento klastr je řízen pomocí technologie Kubernetes. Program využívá ke své správné funkcionalitě kontejnery. Tyto kontejnery jsou tzv. Docker kontejnery. Použité technologie jsou v práci detailně popsány a rozebrány. Dále se práce zabývá rešerží ukládání hesel v současných systémech a tím jak hesla vypadají. Na závěr na klastru bude proveden test různých metod pro lámání hesel. Tyto metody budou popsány a bude analyzováno, jak je klastr efektní a výkonný pro daný typ lámání.

**Klíčová slova** Kubernetes, Ansible, klastr, Docker, distribuované lámání, hesla, hashcat, nasazení.

---

## Abstract

The main goal of the thesis is to setup a cluster managed by kubernetes for password recovery. Next step is to describe used technologies as Docker, Ansible and Hashcat. Thesis contains description of how the passwords are stored and most known attacks to crack them. Successful deployment and password cracking leads to analyzing speed of the cluster and the particular cracking method.

**Keywords** Kubernetes, Ansible, cluster, Docker, distributed cracking, passwords, hashcat, deployment.

---

# Obsah

Úvod	1
<b>1 Cíl práce</b>	<b>3</b>
1.1 Kubernetes	3
1.1.1 Stavební kameny Kubernetes	4
1.1.1.1 Master	5
1.1.1.2 Pod	5
1.1.1.3 Plánovač	5
1.1.1.4 Controller Manager	5
1.1.1.5 API server	5
1.1.1.6 Kubelet	5
1.1.1.7 Kube-Proxy	5
1.1.2 Kubernetes a tajemství	5
1.1.3 Aktualizace obraz	5
1.1.4 Sdílené datové prostory	5
1.1.5 Cloudový poskytovatelé	5
1.2 Docker	5
1.2.1 Kontejner vs. virtuální počítač	5
1.2.2 Stavební kameny Dockeru	6
1.2.2.1 Jmenné prostory	6
1.2.2.2 Kontrolní skupina	7
1.2.2.3 Docker daemon	7
1.2.2.4 Docker klient	7
1.2.2.5 Docker registr	7
1.2.2.6 Obrazy	8
1.2.3 Systémová kontejnery	8
1.2.4 Proč použít Hashcat?	8
<b>2 Hesla</b>	<b>9</b>

2.1	Hašovací funkce . . . . .	9
2.1.1	Vlastnosti hašovací funkce . . . . .	9
2.1.2	Naorzeninový paradox . . . . .	10
2.1.3	Windows . . . . .	10
2.1.4	Linux . . . . .	10
2.1.5	MacOS . . . . .	10
2.2	Útoky na hesla . . . . .	10
2.2.1	Hrubou silou . . . . .	10
2.2.2	Pomocí masky . . . . .	10
2.2.3	Se slovníkem . . . . .	10
2.3	Entropie hesla . . . . .	10
2.4	Ochrana před různými útoky . . . . .	10
<b>3</b>	<b>Závěr</b>	<b>11</b>
	<b>Literatura</b>	<b>13</b>
<b>A</b>	<b>Seznam použitých zkratek</b>	<b>15</b>
<b>B</b>	<b>Obsah přiloženého CD</b>	<b>17</b>

---

# Seznam obrázků

1.1	Komponenty Kubernetes . . . . .	4
-----	---------------------------------	---



---

# Seznam tabulek

1.1	Linuxové jmenné prostory . . . . .	7
-----	------------------------------------	---





---

# Úvod



# Cíl práce

Cílem práce je sestavit Kubernetes klastr, nasazen bude pomocí technologie Ansible a následně na něm bude spuštěn software na lámání hesel. Rychlost a výsledky lámání budou analyzovány s ohledem na délku hesla a použití daného druhu útoku. Budou popsány běžně používané útoky na hesla a způsob jejich ukládání na různých operačních systémech. Technologie, jež budou použity, budou popsány do hloubky nutné k porozumění daného řešení a následné možné replikace pro jiná řešení.

## 1.1 Kubernetes

Jméno Kubernetes pochází z Řecka a znamená to kormidelník. Projekt zložili Joe Beda, Brendan Burns, a Craig McLuckie, ke kterým se rychle připojili inženýři z Googlu, jako Brian Grant a Tim Hockin. Software byl vydán v roce 2014.

Kontejnery jsou perfektní způsob, jak vytvářet aplikace tak, aby byly lehce rozšiřitelné a dobře spravovatelné. Kubernetes nám následně pomáhá k jejich nasazení a škálování. Co všechno tedy Kubernetes dokáže:

- **Service discovery and load balancing:** Kubernetes propojují kontejnery zkrze DNS a nebo jejich IP adresy. Pokud na jeden kontejner jde mnoho požadavků, přesměrují tyto požadavky na jiný a tímto způsobem balancují provoz a zajišťují stabilitu aplikace.
- **Orchestrace úložiště:** Dovolují automaticky připojovat vzdálené nebo lokální úložiště do klastru a zajistit tak konzistenci dat, zálohu a vysokou dostupnost z více míst.
- **Automated rollouts and rollbacks:** Můžeme popsat požadovaný stav pro naše nasazené kontejnery pomocí Kubernetes a ty automaticky a s kontrolou zajistí tento stav. Můžeme například automatizovat vytváření

## 1. CÍL PRÁCE

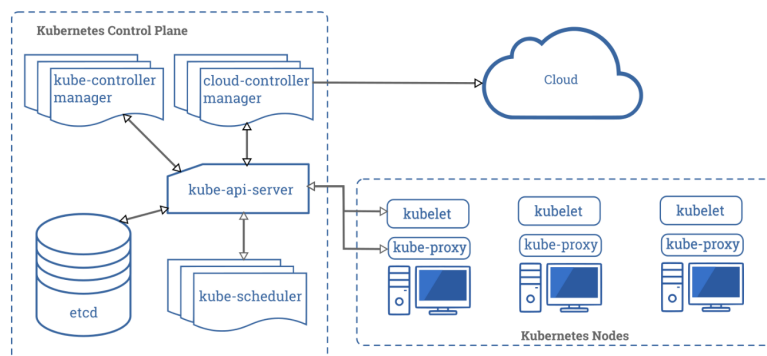
---

nových kontejnerů, smazání starých a převzetí všech zdrojů jako jsou data a další novým kontejnerem.

- **Automatic bin packing:** Předáme Kubernetes vztvořený klastr s uzly a specifikujeme zdroje, které každý kontejner potřebuje pro své správné fungování. Kubernetes se následně postará o nejlepší rozdělení kontejnerů na uzly tak, aby optimalizoval naše zdroje. To se může nejvíce hodit na cloudových řešení, kde se platí za to, kolik se využívá zdrojů.
- **Self-healing:** Kubernetes restartují kontejnery, které selžou, nahradí je, zahodí pokud přestanou odpovídat na specifikované health checky a přestane je nabízet klientům nebo jiným službám dokud nejsou připravené.
- **Secret and configuration management:** Kubernetes napomáhají ukládání a sdílení či měnění citlivých informací zkrze klastr. Při změně těchto informací tedy nemusíme znovu vytvářet kontejnery a nově všechno předělat. A to bez jejich zveřejňování v konfiguracích.

### 1.1.1 Stavební kameny Kubernetes

Aby Kubernetes zajistili funkčnost klastru je zapotřebí rozdělit práci do několika komponent, které se starají o správný chod. Níže je znázorněno, z čeho se skládají. Dále se komponenty popíší a vysvětlí se na nich kompletní funkcionality.



Obrázek 1.1: Komponenty Kubernetes

- 1.1.1.1 Master
- 1.1.1.2 Pod
- 1.1.1.3 Plánovač
- 1.1.1.4 Controller Manager
- 1.1.1.5 API server
- 1.1.1.6 Kubelet
- 1.1.1.7 Kube-Proxy
- 1.1.2 Kubernetes a tajemství
- 1.1.3 Aktualizace obraz
- 1.1.4 Sdílené datové prostory
- 1.1.5 Cloudový poskytovatelé
- 1.2 Docker

Docker je otevřená platforma pro vývoj, dodání a spouštění aplikací. Umožňuje oddělení aplikací od infrastruktury, tedy můžeme dodávat software rychleji a bez problémů, které se váží k různorodosti prostředí, ve kterém aplikace běží. Svou fylozofií jsou velice podobné virtualním počítačům. Rozdíly mezi těmito různými pohledy na věc budou rozebrány dále v textu.

Docker zprostředkovává platformu pro zabalení aplikace i se všemi jejími závislostmi. Izoluje danou aplikaci od ostatních běžících procesů na daném počítači a zajišťuje tak její bezpečí. Docker kontejner je velice nenáročný na hardware, můžeme jich tedy na daném počítači spustit velice mnoho.

Fylosofie kontejnerů je taková, že každý kontejner je odpovědný pouze za jednu danou část aplikace. Pro příklad máme naši webovou aplikaci. Budeme tedy mít alespoň tři docker kontejnery. Jeden na kterém poběží NGINX a bude zprostředkovávat naši aplikaci uživatelům. Další bude mít naši aplikaci a ve třetím poběží databáze.

Konterjnery fungují tedy jako malé počítače, mají izolované veškeré svoje systémové zdroje ( paměť, procesy, internetové rozhraní ). Díky tomuto mohou být rychle a jednoduše přidány, nebo odebrány.

### 1.2.1 Konterjner vs. virtuální počítač

Virtualizace je odpověď na problém různorodých prostředí mezi vývojáři a zákazníky. Problém ,který virtualizace a kontejnerizace především řeší je různorodost prostředí mezi zákazníkem a dodavatelem softwaru. Při jeho předávání dochází ke změně prostředí, jsou nainstalované jiné verze závislostí a operačního systému a aplikace se může chovat neočekávaně.

Podíváme se jak se tyto dvě technologie liší a proč se svět žene právě směrem kontejnerizace, když zde již je řešení.

Virtuální počítač je regulérní stroj, který běží na daném hostovi. Tento stroj má svůj kernel, svůj operační systém a ke zdrojům přistupuje přes tzv. hypervizor např.: QEMU, nebo VirtualBox. Hypervizor zprostředkovává přístup virtuálního stroje k systémovým zdrojům.

Pro to, aby na hostovi, nebo-li na systému, který má nainstalovaný hypervizor mohlo běžet více virtuálních strojů stačí jedna jeho instance. Nevýhoda tohoto řešení je taková, že se mnoho zdrojů duplikuje. Řekněme, že na hostitelském systému poběží tři aplikace. Každá taková aplikace bude izolovaná od ostatních pomocí virtuálního stroje. Dejme tomu, že to bude databáze, webový server a stroj pro vzdáleného uživatele. Níže uvidíme náskok tohoto řešení.

To samé, jako je na obrázku výše se pokusíme realizovat pomocí Dockeru a kontejnerů. Kontejnery, jelikož využívají overlayFS jsou schopny poskytnout jádro operačního systému kontejneru bez zbytečné kopie a využívají copy-on-write funkcionality. Níže uvidíme jak za pomoci jmenných prostorů, kontrolních skupin a overlayFS je tento přístup úspornější a rychlejší než virtualizace.

Místo, které jsme na hostitelském systému ušetřili však není jediná výhoda. Na tomto příkladu se však rozdíl mezi těmito technologiemi vysvětluje nejlépe. Dalšími výhodami je rychlost spuštění kontejneru a virtuálního stroje. Při spuštění se pouze připojí obraz OS, vytvoří se izolované procesy a popřípadě se omezí i zdroje, které má kontejner využívat. Nehledě na to, že pokud kontejner nemá omezení nebo limit využitých zdrojů alokuje si je dynamicky oproti virtuálnímu počítači, který si pro sebe naalokuje danou paměť při spuštění.

### 1.2.2 Stavební kameny Dockeru

Jak je již uvedeno v předešlé kapitole, Docker využívá vychytávky linuxového kernelu pro svojí funkcionalitu. Díky tomuto perfektně funguje na počítačích, kde běží OS založený na Linuxu. V následujících sekcích budou tyto technologie blíže popsány a bude vysvětlena jejich důležitost.

#### 1.2.2.1 Jmenné prostory

Jmenné prostory zastřešují veškeré zdroje systému tak, že každý proces spuštěn v daném prostoru může používat pouze prostředky, které se váží k tomuto prostoru. Každému procesu se to jeví tak, že má svoje vlastní globální prostředky, které mohou vidět i ostatní procesy z jmenného prostoru, ale ne z jiného. V tabulce 1.1 je možné vidět, jaké jmenné prostory lze v Linuxu nalézt.

Při spuštění kontejneru dojde k vytvoření procesu na hostitelském systému. Procesy dostanou od systému nějaké PID a chovají se jako normální procesy.

Tabulka 1.1: Linuxové jmenné prostory

Jméno	Popis
Cgroup	Cgroup root adresář
IPC	Systém pro komunikaci procesů, POSIX fronty
Network	Síťové rozhraní, protocols, ports, etc
Mount	Připojená zařízení
PID	ID procesů
User	Uživatelská ID a ID skupin
UTS	Hostname a NIS doménu

Pokud se však přihlásíme do kontejneru (command: `docker exec -it name bash`) a podíváme se na procesy běžící v daném kontejneru uvidíme, že procesy mají jiná PID a určitě mají i PID=1. Toto nám umožňuje jmenné prostory.

Každý kontejner může mít svůj vlastní souborový systém a svoje síťové rozhraní. Vše co můžeme oddělit mezi hostitelem a kontejnery je uvedeno v tabulce výše.

#### 1.2.2.2 Kontrolní skupina

Je to vlastnost Linuxového kernelu. Jejich hlavní funkcí je limitovat zdroje. V Dockeru se používají protože dovolují sdílet prostředky mezi hostitelským systémem a dalšími kontejnery.

Často dochází k záměně pojmů mezi kontrolními skupinami a jmennými prostory. Znovu to tedy shrňme. Kontrolní skupiny, nebo-li cgroups omezují co můžeme použít a jmenné prostory nebo-li namespaces omezují co jsme schopni vidět v systému.

#### 1.2.2.3 Docker daemon

Docker daemon nebo-li dockerd poslouchá dotazy na docker API a spravuje objekty jakou jsou docker obrazy, kontejnery, síť a úložiště. Komunikuje ale i s dalšími daemony, aby byl schopen řídit službu Docker.

#### 1.2.2.4 Docker klient

Je to primární cesta, jak komunikovat s Dockerem. Když použijeme příkazy, jako jsou "docker run", klient odešle příkazy daemone zmíněného výše.

#### 1.2.2.5 Docker registr

Docker registr je úložiště pro naše Docker obrazy. Bez předchozího nastavení hledá dockerd obrazy, které chceme spustit ve veřejném Docker registru. Ob-

## 1. CÍL PRÁCE

---

razy však mohou být dostupné i lokálně, nebo na nějaké jiné službě, např.: gitlab container registry.

Do styku s registrem přicházíme hlavně ve chvílích, kdy provádíme příkazy `docker pull`, `docker push` a `docker run`. Tyto příkazy vždy potřebují znát obraz, který bude spuštěn jako základní vrstva pro nový kontejner, nebo bude stáhnut na lokální počítač, či nasdílen do registru.

### 1.2.2.6 Obrazy

Můžeme si to představit jako šablonu, na které je spuštěn kontejner. Obraz může být složen z vícero obrazů, nebo z nich vycházet.

Pro vytvoření obrazu je třeba soubor `Dockerfile`. Tento soubor obsahuje jednoduché kroky, které je třeba vykonat pro vytvoření konkrétního obrazu. Např.: jaké použijeme a zveřejníme porty, jaké balíčky chceme ve vytvořeném obrazu mít atd.

Každý příkaz v `Dockerfile` vytvoří na lokálním počítači tzv. vrstvu, kterou při úpravě `Dockerfile` mění nebo předělává pouze pokud byla změněna.

### 1.2.3 Systémová kontejnery

Docker kontejnery nejsou však jediné, které se v produkčním prostředí používají. Patří do tzv. aplikačních kontejnerů. Jejich účel je zpravidla spouštět pouze jeden proces. K takovýmto kontejnerům můžeme ještě přidat kontejner z Rocket. Hlavním rozdílem je to, že rtk nemá na systému spuštěného daemona jako má např. Docker. Při spuštění se tedy pod běžícím spustí další.

Dále tady máme systémové kontejnery. Ty jsou používány jako klasické OS. Na jednom silném stroji může běžet několik takových kontejnerů a ty mohou uživateli poskytovat oddělené prostředí od celého serveru a nabídnout mu izolovaný prostor od ostatních pomocí výše zmíněných technologií. Tuto možnost zastřešuje projekt LXC později LXD.

### 1.2.4 Proč použít Hashcat?



## Hesla

Hesla můžeme vidět všude a ne jen v informatice. Pokud se podíváme zpět do historie např. do doby velkého Caesara a jeho šifry, ke které je třeba znát číslo, o které se posouvají znaky ve zprávě. Jak tedy můžeme vidět, hesla neslouží pouze k naší autentizaci vůči nějaké službě či serveru. Může je také použít k podepsání citlivých dokumentů jako je třeba příloha e-mailu. Následně pak nemůžeme popřít jeho poslání. Tomuto se říká elektronický podpis.

Hesla však mají nejednu nevýhodu. Útočník může s naším nebo i bez našeho vědění odhalit naše heslo a tím nám narušit naše soukromý. Hesla mohou také být v systémech, které používáme uložena nepatřičným způsobem, jako je například čistý text bez použití žádných ochranných prostředků.

Hesla též mohou ze systému uniknout. V tomto případě, pokud byla hesla uložena neptřičným způsobem nemusí se potenciální útočník nějak přemáhat, aby uživatele kompromitoval. Proto se zaměříme na to jak mohou a jak skutečně jsou uložena v nejpoužívanějších systémech.

### 2.1 Hašovací funkce

Jsou to takové funkce  $f: X \rightarrow Y$ , pro něž je snadné z jakékoli hodnoty  $x \in X$  vypočítat  $y = f(x)$ , ale pro náhodně vybraný obraz  $y \in f(X)$  nelze v relevantním čase najít její vzor  $x \in X$  tak, aby  $y = f(x)$ .

Přitom víme, že takový vzor existuje nebo jich existuje dokonce velmi mnoho. To kolik jich existuje se odvíjí jakou hašovací funkci použijeme.

#### 2.1.1 Vlastnosti hašovací funkce

Abychom mohli funkci považovat za hašovací, musí mít následující vlastnosti:

- jakékoliv množství vstupních dat poskytuje stejně dlouhý výstup (otisk),
- malou změnou vstupních dat dosáhneme velké změny na výstupu,

## 2. HESLA

---

- z hashe je prakticky nemožné rekonstruovat původní text zprávy,
- v praxi je vysoce nepravděpodobné, že dvěma různým zprávám odpovídá stejný hash, jinými slovy pomocí hashe lze v praxi identifikovat právě jednu zprávu (ověřit její správnost).

### 2.1.2 Naorzeninový paradox

### 2.1.3 Windows

Windows se chovají jinak v doméně a jinak mimo ní. Pokud je počítač v doméně je preferován autentizační protokol kerberos. V současných Windows Server edicích je implementován Kerberos verze 5. Kerberos v základní nastavení operuje na portu 88 a k šifrování používá symetrickou šifru. Pokud počítač není nastaven aby se autentikoval pomocí protokolu Kerberos používají Windows šifrování NTLM.

### 2.1.4 Linux

Hesla v linuxových systémech se skládají ze dvou konkrétních souborů.

/etc/shadow - obsah a strukturu toho souboru můžeme vidět na následujícím obrázku.

/etc/passwd - obsah a strukturu tohoto souboru můžeme vidět na následujícím obrázku.

V /etc/shadow jsou hesla uložena pomocí hashe.

### 2.1.5 MacOS

## 2.2 Útoky na hesla

### 2.2.1 Hrubou silou

### 2.2.2 Pomocí masky

### 2.2.3 Se slovníkem

## 2.3 Entropie hesla

## 2.4 Ochrana před různými útoky

## **Závěr**

[1]1



---

## Literatura

- [1] D. Merkel, “Docker: lightweight linux containers for consistent development and deployment,” *Linux journal*, vol. 2014, no. 239, p. 2, 2014.



## Seznam použitých zkratk

**GUI** Graphical user interface

**XML** Extensible markup language





## Obsah přiloženého CD

	readme.txt.....	stručný popis obsahu CD
	exe .....	adresář se spustitelnou formou implementace
	src	
	impl.....	zdrojové kódy implementace
	thesis .....	zdrojová forma práce ve formátu L <sup>A</sup> T <sub>E</sub> X
	text .....	text práce
	thesis.pdf .....	text práce ve formátu PDF
	thesis.ps .....	text práce ve formátu PS