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 principu 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 předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, 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 zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). 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 zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. března 2020 . . . . . .

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

Seznam obrázků

Seznam tabulek

Úvod

Kapitola 1

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 následné pomáhá k jejich nasazení a škálování. Co všechno tedy Kubernetes dokáží:

- **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

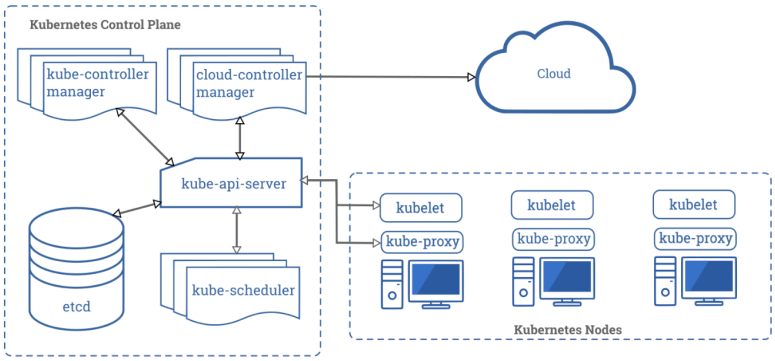
nových kontejnerů, smazání starých a převzetí všech zdrojů jako jsou data a další novým kontejnerem.

- **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

V momentě kdy nasadíme Kubernetes, dostaneme klastr, který se skládá z výpočetních uzlů. Takovému uzlu se říká node a na něm se spouští kontejnerizované aplikace. Každý klastr se skládá z alespoň jednoho takového nodu.

Tyto výpočetní nody poskztují prostor pro pody. Pod si můžeme představit jako balík, ve kterém jsou umístěné kontejnery. V praxi je nejčastěji Control Plane rozdělen na více počítačů, aby se zajistila vysoká dostupnost a maximalizovala se odolnost proti chybám.

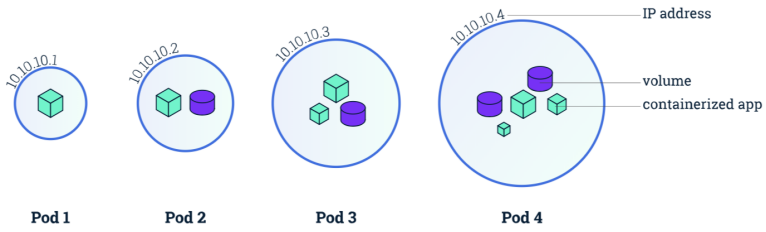


Obrázek 1.1: Komponenty Kubernetes

1.1.1.1 Pod

Pod je základní stavební jednotka Kubernetes. Je to balíček kontejnerů nebo též samostatný kontejner, který je specifikován v deploymentu a Kubernetes se

o něj mají starat. Pro představu je zde obrázek, z čeho se pod skládá. Takových podů může být na jednom nodu několik. To jak se rozhodneme zabalit naši aplikaci do podů je čistě na tom, jak se rozhodneme. Abychom však neporušovali konvence tzv. service architektury. Musíme do jednoho podu dávat služby pouze takové, které dávají smysl. Pro příklad nebudeme do jednoho podu balit databázi s webovým UI. Při škálování by se totiž nejen že replikovala aplikace ale i databáz.



Obrázek 1.2: Z čeho se skládá pod.

1.1.1.2 Control Plane

Součásti Control plane řídí a rozhodují co se stane s klastrem, např. řídí plánování, detekci a odpovědi klastru na události, jako jsou start nového podu pokud není splněn požadovaný stav.

Komponenty Control plane mohou být spuštěny na jakémkoliv počítači v klastru. Pro jednoduchost jsou však skripty napsané tak, aby tyto komponenty byly spuštěny na jednom počítači. Samotná aplikace je pak směřována mimo tento počítač.

- **kube-apiserver**: Tato komponenta poskytuje API uživateli. Je to front end pro Control plane. Hlavní implementací je kube-apiserver, ten je navržen, aby se dokázal horizontálně škálovat, to znamená, že se dokáže replikovat do více instancí. Můžeme tedy spustit více instancí a balancovat provoz.

- **etcd**: Skládá se z vysoce dostupné databáze s klíčem a hodnotou pro Kubernetes a všechna klastrová data. Její důležité vlastnosti jsou: jednoduchost, bezpecnost, rychlost, spolehlivost.

- **kube-scheduler**: Tato komponenta kontroluje a vytváří nové pody, pro které zatím nebyl přiřazen žádný node. Je odpovědná za přiřazení takového podu na nějký node podle požadavků. Vždy musí brát v potaz zdroje, které jsou nutné, hardwarové a softwarové vazby, affinity a

anti-affinity specifikace, kde jsou uložena data, vnitřní vytížení a časové termíny.

- **kube-controller-manager**: V této komponentě je spuštěn proces pro kontrolu klastru. Skládá se z více částí, které jsou zkompilovány do jednoho programu aby mohly běžet v jednom procesu, který se o vše stará. A stará se o následující:

  1. **Node Controller**: Odpovědný za upozonění pokud jeden z nodů přestane odpovídat.
  2. **Replication Controller**: Odpovědný za dodržování správného počtu spuštěných aplikací v klastru. Pokud tedy část aplikace spadne je odpovědný za start nové instance.
  3. **Endpoints Controller**: Přidá nová uzly, nebo-li připojí služby a pody do klastru po jejich spuštění.
  4. **Service Account a Token Controllers**: Vytváří výchozí účty a přístupové tokeny k API pro nové namespacy.

- **cloud-controller-manager**: Tento daemon dovoluje připojit cloudové služby. Byl vytvořen pro možnost oddělení cloudových vývojařů od zdrojového kódu Kubernetes. cloud-control-manager je možné připoji k jakému koliv poskytovateli, který implementuje cloudprovider.interface.

1.1.2 Komponenty nodu

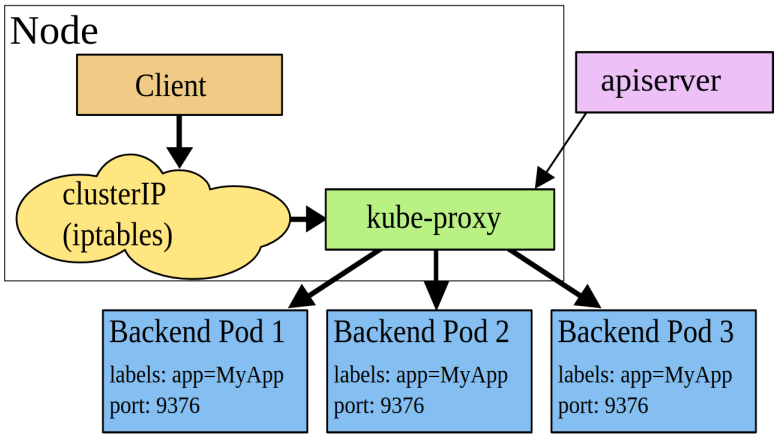
Node, jak je již uvedeno výše, je v podstatě stroj, který je připojen do klastru. Na tomto nodu mohou běžet pody, což je balík kontejnerů. Níže je popsáno z čeho se tento node skládá a co na něm musí běžet, aby Kubernetes správně operovalo.

1.1.2.1 Kubelet

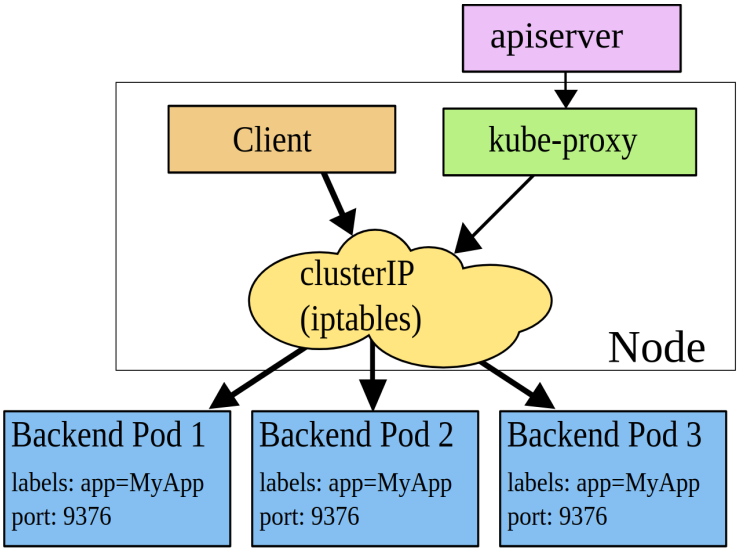
Je agent zajišťující běh všech kontejnerů v daném podu. Kubelet převezme množinu požadavků na daný pod, tzv. PodSpecs a zajistí, že všechny kontejnerz popsány v těchto specifikacích jsou spuštěné a odpovídají na health checky.

1.1.2.2 Kube-Proxy

Je síťová proxy, která běží na každém nodu. Spravuje síťová pravidla. Zajišťuje tak komunikaci mezi jednotlivými pody a nody uvnitř klastru. Používá szstémovou vrstvu na filtrování paketů, pokud je dostupná, jinak forwarduje provoz za pomoci svých prostředků. Níže znázorněno na obrázcích.



Obrázek 1.3: Kubernetes proxy - User space proxy mode



Obrázek 1.4: Kubernetes proxy - iptables proxy mode

1.2 Ansible

Ansible je automatizační nástroj pro konfiguraci systému, nasazení softwaru, aktualizac. Jeho nejsilnější stránka je nulové výpadky systému při aktualizaci balíčků, nebo automatické nastavovat dané zařízení.

Jeho hlavními cíly jsou jednoduchost a nenáročnost. Kód by měl být čitelný i pro lidi, kteří nejsou obeznámeni s programem. Je schopen pokrýt různě velké prostředí od malých podniků až po velice obsáhlou infrastrukturu.

Ansible se připojí na vzdálený počítač pomocí OpenSSH pomocí uživatele, který je současně přihlášen. Na spravovaném počítači není třeba žádný agent.

Je možnost nakonfigurovat Ansible, aby pro připojení nepoužíval OpenSSH, ale i kerberos nebo LDAP.

Ansible a jemu podobné nástroje se použijí v případě toho, že máme více serverů, nebo pokud budeme dodržovat trend IaaS. Kdyby měl admin ve firmě nasazovat nebo aktualizovat například sto počítačů, tak na každém stráví 15 minut. To bude 1500 minut a to je 25 hodin. Proto použije Ansible a za 20 minut je hotov.

Pro příklad budeme instalovat Docker na 3 počítače. Na těchto počítačích budeme mit v souboru

keys naše veřejné klíče pro ssh. Na hostovi, ze kterého budeme instalovat je potreba Ansible. Tedy sudo apt install ansible. V souboru hosts, který je pro Ansible tzv. inventářem naspecifikujeme stanice, na které budeme instalovat Docker.

1.2.1 Komponenty

1.2.1.1 Control node

Jakýkoliv počítač s nainstalovaným Ansible a pythonem, může spouštět příkazy nebo tzv. playbooky. Tento počítač se nazývá control node. Takových můžeme mít klidně více, ne však počítače, které mají nainstalovaný operační systém Windows.

1.2.1.2 Managed node

Je jakékoliv síťové zařízení. Managed nodes můžeme také nayývat jako hosts. Tyto zařízení nemusejí mít nainstalovaný Ansible, ale musejí mít nainstalovaný python. Ansible může být nakofigurován, aby používal specifikovanou verzi pythonu, pokud není specifikována, spustí se na hostu jeho defaultní.

1.2.1.3 Iventory

Je seznam všech nastavovaných zařízení. Často se nazývá hostfile. v tomto souboru nastavujeme skupiny zařízení, jejich IP adresy a další specifikace, například jaký python má daný host použít.

1.2.1.4 Modules

Jsou to jednotlivé části kódu, které bude Ansible spouštět. Každý modul má speciální použití. Vše od správy uživatelů (user) přes nastavení systému (systemd) až k instalovaní balíčků (apt, yum). Můžeme spustit jeden modul v tasku, nebo více v playbooku. Pro přehlednost neuvádím všechnz možné moduly, jelikož je jich přes tři tisíce.

1.2.1.5 Tasks

Jsou jednotky, které se museji provést. Nejčasteji specifikované v deployment souboru.

1.2.1.6 Playbooks

Je seřazený seznam tasků, které se musí vykonat. Ničemu neuškodí pokud se plazbook spustí znovu, protože Asnible skontroluje stav daného tasku. Playbooky jsou psané podle konvenci YAMLu.

1.3 Docker

Docker je otevřená platforma pro vývoj, dodání a spouštění aplikací. Umožnuje 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 jejímy 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 naší webovou aplikaci. Budeme tedy mít alespoň tři docker kontejnery. Jeden na kterém poběží NGINX a bude zprostředkovávat naší aplikaci uživatelům. Další bude mít naší 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.

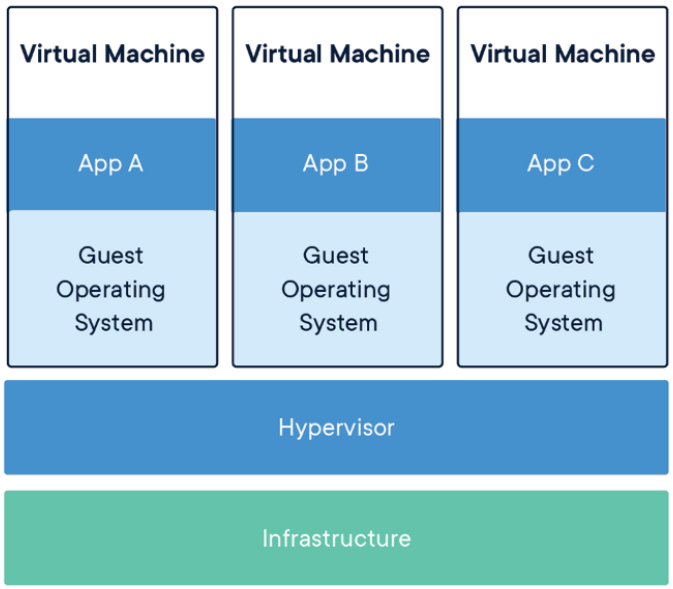
1.3.1 Konterjner vs. virtuální počítač

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.

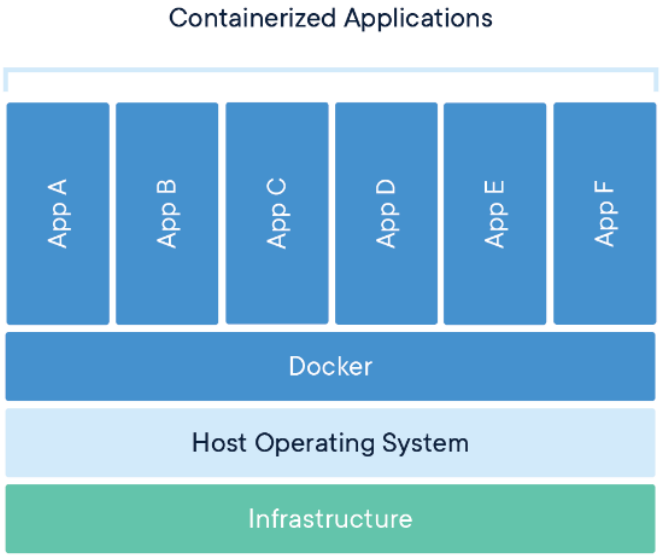
Pro to, aby na hostovi, nebo-li na systému, který má nainstalovaný hypervizor mohlo běžet více virtuálních strojů stačí jedna jeho instance. Nevýhoda tohoto řešení je taková, že se mnoho zdrojů duplikuje. Řekněme, že na hostitelském systému poběží tři aplikace. Každá taková aplikace bude izolovaná od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



Obrázek 1.5: Oddělení aplikací za pomoci hypervizoru a virtuálních strojů.

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 rozdíly mezi těmito technologiemi vysvětlují 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.



Obrázek 1.6: Oddělení aplikací za pomoci Dockeru

1.3.2 Stavební kameny Dockeru

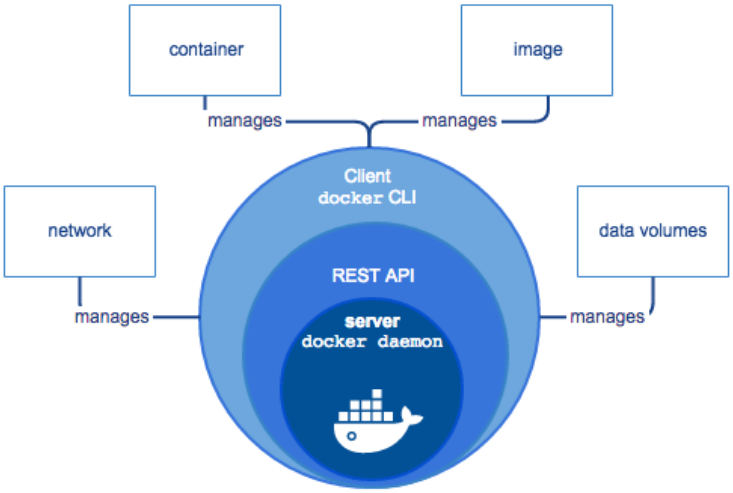
Jak je již uvedeno v předešlé kapitole, Docker využívá vychytávky linuxového kernelu pro svojí funkcionalitu. Díky tomuto perfektně funguje na počítačích, kde běží OS založený na Linuxu. V následujících sekcích budou tyto technologie blíže popsány a bude vysvětlena jejich důležitost.

1.3.2.1 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 videt, 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. 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 umožňují 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.



Obrázek 1.7: Docker a jeho komponenty.

Tabulka 1.1: Linuxové jmenné prostory

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

1.3.2.2 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 dalšímy kontejnery.

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

1.3.2.3 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.

1.3.2.4 Docker klient

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

1.3.2.5 Docker registr

Docker registr je úložiště pro naše Docker obrazy. Bez předchozího nastavení hledá dockerd obrazy, které chceme spustit ve veřejném Docker registru. Obrazy 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 příkazý docker pull, docker push a docker run. Tyto příkazy vždy potřebují znát obraz, který bude spouštěn jako základní vrstva pro nový kontejner, nebo bude stáhnut na lokání počítač, či nasdílen do registru.

1.3.2.6 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 Dockerfilu vytvoří na locálním počítači tzv. vrstvu, kterou při úpravě Dockerfilu mění nebo předělává pouze pokud byla změněna.

1.3.3 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 kontejnerz 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 používané 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 ostattních pomocí výše zmíněných technologií. Tuto možnost zastřešuje projekt LXC později LXD.

1.3.4 Proč použít Hashcat?

Kapitola 2

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 potencionální 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 ® Y, pro něž je snadné z jakékoli hodnoty x Î X vypočítat y = f(x), ale pro náhodně vybraný obraz y Î f(X) nelze v relevatním čase najít její vzor x Î 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,

- 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 nastavenim 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 konkretních 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 uložená 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

Kapitola 3

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.

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

XML Extensible markup language

Příloha **B**

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 LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS