Sigurnosni aspekti sustava za orkestraciju Kubernetes

Golubić, Tomislav

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Tomislav Golubić

SIGURNOSNI ASPEKTI SUSTAVA ZA ORKESTRACIJU KUBERNETES

SPECIJALISTIČKI RAD

Zagreb, 2023. godina

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Tomislav Golubic

SECURITY ASPECTS OF THE KUBERNETES ORCHESTRATION SYSTEMS

SPECIALIST THESIS

Zagreb, 2023.

Specijalistički rad izrađen je na Sveučilištu u Zagrebu, Fakultetu elektrotehnike i računarstva u sklopu poslijediplomskog specijalističkog studija „Informacijska sigurnost“.

Mentor: izv. prof. dr. sc. Marin Vuković

Specijalistički rad ima: 174 stranice

Specijalistički rad br.: _____

Povjerenstvo za ocjenu u sastavu:

1. izv. prof. dr. sc. Miljenko Mikuc – predsjednik
2. izv. prof. dr. sc. Marin Vuković - mentor
3. izv. prof. dr. sc. Toni Perković – Sveučilište u Splitu
Fakultet elektrotehnike, strojarstva i brodogradnje - član

Povjerenstvo za obranu u sastavu:

1. izv. prof. dr. sc. Miljenko Mikuc – predsjednik
2. izv. prof. dr. sc. Marin Vuković - mentor
3. izv. prof. dr. sc. Toni Perković – Sveučilište u Splitu
Fakultet elektrotehnike, strojarstva i brodogradnje - član

Datum obrane: 26. travnja 2023.

SAŽETAK SPECIJALISTIČKOG RADA

Složenost današnjih arhitektura računalnih i mrežnih sustava iziskuje velike napore pri administraciji i orkestraciji. Posljednjih godina ponuđeno je više rješenja koja imaju za cilj pojednostavniti postavljanje, upravljanje i nadzor takovih sustava. Prije svega, mogućnost kontejnerizacije te dinamičko upravljanje kontejnerima omogućuju znatno veću fleksibilnost sustava u smislu dostupnosti, skaliranja i upravljanja sigurnošću. Jedno od najraširenijih rješenja za ovu svrhu jest sustav za orkestraciju Kubernetes.

Kubernetes omogućuje orkestraciju kontejnera u klasterima koji se sastoje od više čvorova. Ovo je vrlo važno ne samo za početnu implementaciju, već i za kasnije upravljanje sustavom sa većim brojem kontejnera kao jednim entitetom s ciljem što boljeg skaliranja, dostupnosti, a posebice sigurnosti.

S obzirom na modele prijetnji, a iz perspektive sustava Kubernetes kao vrlo složenog sustava, specijalistički rad usmjerio se na osnovna područja sigurnosnih značajki sustava za orkestraciju Kubernetes.

U specijalističkom radu opisan je način rada, osnovne konfiguracijske postavke sustava za orkestraciju Kubernetes, arhitektura sustava te komponente i objekti sustava. Nadalje, u radu se pokušalo dati odgovore na neka od pitanja, što su Kubernetes, čemu služe i zašto su potrebni te kako osigurati što višu razinu sigurnosti sustava za orkestraciju Kubernetes.

U radu su razmatrane samo one sigurnosne kontrole koje se nalaze unutar neposrednog okruženja sustava za orkestraciju Kubernetes i to prvenstveno iz perspektive sistemskog održavanja sustava za orkestraciju Kubernetes.

U praktičnom dijelu rada prikazane se sigurnosne konfiguracijske postavke sustava za orkestraciju Kubernetes s naglaskom na područja koja se odnose na autentifikaciju, autorizaciju i kontrolu pristupa temeljenu na ulogama (RBAC), izolaciju resursa, sigurnost čvorova, sigurnost mreže, sigurnost kontejnera i pripadajućih repozitorija, nadzor, zapisivanje i reviziju te periodična ažuriranja sustava Kubernetes.

U specijalističkom radu sigurnosne postavke sustava za orkestraciju Kubernetes opisane su i prikazane kroz primjere na postavljenom sustavu koji se sastoji od klastera sa tri čvora.

Ključne riječi: Kubernetes, kontejner, mikroservis, orkestrator, klaster, čvor, pod, servis, kublet, API, proxy, yaml, rbac, secret, autorizacija, autentifikacija, aplikacija, sigurnosne kontrole, certifikati, DNS, sigurnosna politika, mreža, nadzor, zapisivanje, revizija, Prometheus, Grafana, Elasticsearch, Lens, ažuriranja, nadogradnje, CVE, Kube Bench, Kubescape.

SUMMARY OF SPECIALIST THESIS

The complexity of today's computer and network system architectures requires great efforts in administration and orchestration. In recent years, several solutions have been offered that aim to simplify the setup, management and monitoring of such systems. First of all, the possibility of containerization and dynamic management of containers enable significantly greater flexibility of such systems in terms of availability, scaling and security management. One of the most widespread solutions for this purpose is the Kubernetes orchestration system.

Kubernetes enables the orchestration of containers in clusters consisting of multiple nodes. This is very important not only for initial implementation, but also for later management of a system with a large number of containers as a single entity with the aim of better scaling, availability, and especially security.

Considering the threat models, and from the perspective of the Kubernetes system as a very complex system, the specialist work focused on the core areas of the security features of the Kubernetes orchestration system.

The specialist paper describes the mode of operation, basic configuration settings of the Kubernetes orchestration system, system architecture, and system components and objects. Furthermore, the paper tried to provide answers to some of the questions, what are Kubernetes, what are they for and why are they needed, and how to ensure the highest level of security of the Kubernetes orchestration system.

The specialist thesis considered only those security controls that are located within the immediate environment of the Kubernetes orchestration system, primarily from the perspective of system maintenance.

In the practical part of the work, the security configuration settings of the Kubernetes orchestration system are presented with an emphasis on areas related to authentication, authorization and role-based access control (RBAC), resource isolation, security of nodes, network security, security of containers and associated repository, monitoring, logging and auditing, and periodic updates to the Kubernetes system.

In the specialist thesis, the security settings of the Kubernetes orchestration system are described and shown through examples on a set system consisting of a three-node cluster.

Keywords: Kubernetes, container, microservice, orchestrator, cluster, node, Pod, service, kublet, API, proxy, yaml, rbac, secret, authorization, authentication, application, security controls, certificates, DNS, security policy, network, monitoring, logging , audit, Prometheus, Grafana, Elasticsearch, Lens, updates, upgrades, CVE, Kube Bench, Kubescape.

SADRŽAJ SPECIJALISTIČKOG RADA:

1. UVOD 6

2. SUSTAV KUBERNETES KAO KONTEJNER ORKESTRATOR 12

2.1 Zašto nam je potreban sustav Kubernetes? 14

2.2 Varijante sustava Kubernetes 18

2.3 Način rada, arhitektura i komponente sustava Kubernetes 24

2.4 YAML programski jezik 42

2.5 Instalacije sustava Kubernetes 45

3 SIGURNOSTI SUSTAVA ZA ORKESTRACIJU KUBERNETES 47

3.1 Modeli prijetnji u sustavu Kubernetes 52

3.2 Načelo najmanje privilegije 66

3.3 Autentifikacija, autorizacija i RBAC model 68

3.4 Sigurnost Kubernetes klaster komponenti 78

3.5 Sigurnost kontejnera i Pod-ova u sustavu Kubernetes 88

3.6 Sigurnost *Secret* objekata 99

3.7 Sigurnost mreže sustava Kubernetes 103

4 NADZOR SUSTAVA KUBERNETES 112

4.1 Ključne metrike u praćenju sustava Kubernetes 112

4.2 Alati za nadzor Kubernetes klastera 115

5 ZAPISIVANJE U SUSTAVU KUBERNETES 125

5.1 Kako funkcionira zapisivanje u sustavu Kubernetes 125

5.2 Alati za zapisivanje sustava Kubernetes - prikupljanje i analiza zapisa 129

6 REVIZIJA SUSTAVA KUBERNETES 132

7 AŽURIRANJA I NADOGRADNJE SUSTAVA KUBERNETES 139

8 KUBE BENCH SIGURNOSNO TESTIRANJE 141

9 KUBE SCAPE SIGURNOSNO TESTIRANJE 161

10 ZAKLJUČAK 168

11 ŽIVOTOPIS AUTORA 171

12 BIOGRAPHY OF THE AUTHOR 172

13 POPIS LITERATURE 173

14 POPIS INTERNETSKIH IZVORA 174

1. UVOD

Naziv "Kubernetes" dolazi od grčke riječi koja znači kormilar - osoba koja upravlja brodom. Ova tema ogleda se i u logotipu projekta, koji je predstavlja kormilo broda sa sedam krakova.

Slika 1. Logo sustava za orkestraciju Kubernetes (Izvor: https://kubernetes.io/)

Neki od članova tima koji su sudjelovali u stvaranju sustava Kubernetes htjeli su ga nazvati „Seven of Nine“. „Seven of Nine“ je Borg iz serijala „Star Trek“ kojeg je spasila posada „USS Voyagera“ pod zapovjedništvom kapetanice Kathryn Janeway.

Nažalost, zakoni o autorskim pravima spriječili su da se nazove „Seven of Nine“. Ali, kreatori sustava Kubernetes logotipu su dali sedam krakova kao suptilnu referencu na „Seven of Nine“.

Sustav Kubernetes često će se nazivati i skraćeno kao "K8s" (izgovara se "kejts"). Broj 8 zamjenjuje 8 znakova između slova "k" i "s" i zato se ljudi ponekad šale da Kubernetes ima djevojku po imenu Kate.

Kubernetes je rješenje za upravljanje kontejnerima otvorenog koda koje je Google prvobitno najavio 2014. godine. Nakon njegovog početnog izdanja u srpnju 2015. godine, Google je donirao sustav Kubernetes Zakladi CNCF (eng. Cloud Native Computing Foundation).

Slika 2. Cloud Native Computing Foundation logo (Izvor: https://www.cncf.io/)

Izvorni kreatori projekta tj. sustava Kubernetes bili su Joe Beda, Brendan Burns i Craig McLuckie, no ubrzo pridružili su im se i drugi inženjeri iz Google-a kao što su Brian Grant i Tim Hockin.

Što je sustav Kubernetes? Sedam (pitate se zašto baš 7?) brzih činjenica o sustavu Kubernetes možemo pročitati (i čuti) direktno od dvojice osnivača/pokretača sustava na slijedećem linku: https://news.vmware.com/technologies/what-is-kubernetes-7-facts.

No, nije na odmet na samom početku navesti tih sedam (7) brzih činjenica o kojima će se, između ostaloga, detaljnije govoriti u narednim poglavljima ovog specijalističkog rada, a to su:

1. Kubernetes projekt je pokrenut u Google-u,
2. Kubernetes je danas projekt pod „Cloud Native Computing Foundation (CNCF)“,
3. Kubernetes omogućuje izvorni razvoj u oblaku (eng. cloud-native development),
4. Kubernetes je jedan od najboljih projekata otvorenog koda na GitHub-u,
5. Kubernetes podržavaju svi vrhunski pružatelji usluga u oblaku,
6. Kubernetes je dostupan u raznim varijantama za razne vrste poslovnih okruženja,
7. Broj sedam (7) ima posebno značenje za sustav Kubernetes.

Za razvojnog programera, sustav Kubernetes pruža upravljivo okruženje za izvršavanje, postavljanje, izvođenje, upravljanje i orkestriranje kontejnera u klasterima ili na klaster čvorovima.

Za DevOps inženjere i sistem administratore, Kubernetes pruža kompletan sustav koji omogućuje automatizaciju mnogih operacija za upravljanje razvojnim, testnim i produkcijskim okruženjima.

Orkestracija kontejnera omogućuje koordinaciju kontejnera u klasterima koji se sastoje od više čvorova (eng. Nodes). Ovo je vrlo važno ne samo za početnu implementaciju, već i za upravljanje većim brojem kontejnera kao jednim entitetom u svrhu skaliranja, dostupnosti i sl..

Kubernetes klasteri mogu se instalirati na razne javne i privatne oblake (AWS, Google Cloud, Azure, Linode) kao i na fizičke poslužitelje u privatnim podatkovnim centrima.

Osim toga, Google Container Engine može pružiti distribuirani Kubernetes klaster. To čini Kubernetes sličnim Linux Kernelu, koji osigurava dosljednost na različitim hardverskim platformama i koja radi na gotovo svakom operativnom sustavu.

U svojoj srži, Kubernetes je orkestrator aplikacija na razini mikroservisa. No, to je puno detalja u jednoj rečenici. Dakle, na početku, razjasnimo osnovne postavke kao što su „Što su mikroservisi?“ te „ Što znači pojam „Cloud-native“? O pojmu „kontejner orkestrator“ govoriti će se u slijedećem poglavlju.

Što su mikroservisi?

U ne tako davnoj prošlosti razvijale su se i implementirale monolitne aplikacije. To je izraz za aplikaciju u kojoj je svaka značajka u paketu kao jedan veliki paket. Klasični izgled takovih aplikacija sastoji se o npr. web prednjeg dijela aplikacije (eng. Web Frontend), dijela aplikacije za provjeru autentičnosti i logiranje, sustava za pohranu podataka, sustava za izvješćivanje i sl. sve čvrsto povezano kao jedna cjelina tj. aplikacija. To znači da, ako se želi promijeniti jedan dio aplikacije, mora se promijeniti gotovo sve. Ovakva situacija je vrlo nezgodna i daleka od idealnog.

S druge strane, aplikacija na razini mikroservisa ima isti skup značajki – web sučelje, autentifikaciju, logiranje, pohranu podataka, sustav izvješćivanja i sl. i svaku od njih tretira kao vlastitu malu aplikaciju. Odatle dolazi i pojam "mikroservis".

Najčešći obrazac razvoja aplikacija na nivou mikroservisa je razvoj i implementacija svake mikrousluge kao vlastitog kontejnera. Na primjer, kontejner za mikroservis web prednjeg dijela aplikacije, drugi kontejner kao mikroservis za provjeru autentičnosti, mikroservis za izvješćivanje itd. Svaki je neovisan i povezan preko mreže kako bi se stvorio isti rezultat.

Mikrousluge (mikroservisi) povezane su dizajnom, a to je temeljna odlika ili karakteristika za mogućnost promjene jedne mikrousluge bez utjecaja na ostale. Tehnički govoreći, svaki mikroservis obično izlaže API (eng. Application Programming Interface) preko IP mreže koju drugi mikroservisi koriste.

Postoje i druge prednosti obrasca dizajna mikroservisa, a to je npr. razvoj značajki kao neovisnih mikroservisa koji omogućuje njihovo razvijanje, implementaciju, ažuriranje, skaliranje ... , bez utjecaja na druge dijelove aplikacije.

Međutim, mikrousluge imaju i neke ne tako dobre strane, a to su npr. da takova arhitektura sustava ima puno dijelova kojima upravljaju različiti timovi i iz tog razloga sve zajedno zahtijeva vrlo pažljivo upravljanje.

Konačno, ova dva načina projektiranja aplikacija – monolitni naspram mikroservisa – nazivaju se obrascima dizajna. Obrazac dizajna mikrousluga je najčešći model u trenutnoj eri dizajna aplikacija u oblaku.

Kao što je već i rečeno, svaki od tih pojedinačnih usluga naziva se mikroservis. Velika je vjerojatnost da je svaki od njih kodiran i u „vlasništvu" drugog razvojnog tima. Svaki od njih može imati svoj vlastiti razvojni ciklus i može se samostalno skalirati. Na primjer, može se skalirati mikrousluga „košarice" u nekoj e-trgovini bez utjecaja na bilo koji drugi dio sustava. Izrada aplikacija na ovaj način ključna je za značajke koje su karakteristične za Cloud Native aplikacije.

Što znači pojam „Cloud Native" ?

Aplikacija koja se nalazi u oblaku mora zadovoljavati slijedeće preduvjete:

- Skalabilna na zahtjev (eng. Scale-on-Demand)
- Mogućnost „samoizliječenja" (eng. Self-Heal)
- Podrška za ažuriranja tijekom rada servisa (eng. Support-Rolling-Updates)
- Pokretanje servisa bilo gdje, gdje je postavljen sustav Kubernetes (ili sličan sustav)

Razjasnimo što zapravo znače neki od tih pojmova. Skaliranje na zahtjev je mogućnost da aplikacija i povezana infrastruktura i/ili resursi automatski rastu i smanjuju se kako bi zadovoljili trenutne zahtjeve od strane korisnika ili nekog drugog sustava.

Na primjer, aplikaciju za e-trgovinu možda će trebati povećati resurse aplikacije tijekom npr. blagdanskih razdoblja kada je veća potražnja od strane krajnjih kupaca, a zatim ih smanjiti kada blagdansko vrijeme završi tj. kada potražnja za određenim proizvodima i/ili uslugama padne.

Ako je ispravno konfiguriran, sustav će automatski povećati resurse kada se potražnja poveća. Također, može ih smanjiti kada potražnja padne. Ta mogućnost, ne samo da pomaže tvrtkama da brže reagiraju na neočekivane (ili očekivane) promjene, već i smanjuje troškove održavanja infrastrukture, aplikacija i sustava u cjelini. Ta karakteristika pogotovo je interesantna ukoliko se sustav nalazi u oblaku koji se, naravno, plaća na razini trenutno zakupljenih resursa!

Sustav Kubernetes može „samoizliječiti“ aplikacije i pojedinačne mikroservise. No, to zahtijeva ipak malo više znanja o sustavu Kubernetes. Ali za sada, pojednostavljeno rečeno, kada implementiramo aplikaciju u Kubernetes, „kažemo“ sustavu Kubernetes kako bi ona trebala izgledati tj. detalje kao što su npr. koliko instanci svakog mikroservisa trebamo postaviti i na koje mreže se isti trebaju priključiti.

Kubernetes to sprema kao željeno stanje i prati aplikaciju kako bi se uvjerio da uvijek odgovara željenom stanju. Ako se nešto promijeni, možda se mikroservis sruši, Kubernetes to primijeti i pokrene zamjenu. To se karakteristika naziva i samoizlječenje ili otpornost.

Podrška za ažuriranja tijekom rada servisa je mogućnost ažuriranja dijelova aplikacije bez isključivanja cjelokupne usluge i potencijalno bez da klijenti to primijete što je vrlo korisna karakteristika pogotovo iz perspektive korisničkog zadovoljstva.

Biti „Cloud Native“ nema gotovo nikakve veze isključivo sa javnim oblakom. To je skup značajki i mogućnosti koje su navedene. Kao takva, aplikacija „izvorna u oblaku“ (eng. Cloud Native) može se pokretati bilo gdje, gdje su postavljen sustav Kubernetes – to može biti na Amazon EKS, Azure AKS, Google GKE, Linode, Digital Ocean… u lokalnom podatkovnom centru ili pak na Raspberry Pi.

Ukratko, aplikacije izvorne u oblaku (eng. Cloud Native) otporne su, automatski se skaliraju i mogu se ažurirati bez zastoja. Također, mogu se pokrenuti gdje god postoji postavljen sustav Kubernetes.

Važno je još jedno napomenuti da aplikacije koje su „Cloud Native“ nisu aplikacije koje će se izvoditi samo u javnom oblaku. Da, apsolutno se mogu izvoditi na javnim oblacima, ali također mogu raditi bilo gdje, gdje imamo Kubernetes. Dakle, cloud-native se odnosi na način na koji se aplikacije ponašaju i/ili reagiraju na određene događaje ili stanja.

U slijedećem poglavlju, **2. Sustav Kubernetes kao kontejner orkestrator** opisati će se sustav Kubernetes kao kontejner orkestrator, zašto nam je potreban sustav Kubernetes, varijante sustava Kubernetes, način rada, arhitekturu i komponente (objekte), osnove YAML programskog jezika te na kraju opisati neke od varijanti i načina instalacije sustava Kubernetes.

2. SUSTAV KUBERNETES KAO KONTEJNER ORKESTRATOR

Kubernetes (K8s) je sustav otvorenog koda za automatizaciju implementacija, skaliranja i upravljanja kontejnerskim aplikacijama. Jednom riječju, sustav za orkestraciju kontejnera.

Orkestracija je automatizirano upravljanje i koordinacija računalnih sustava, aplikacija i usluga. Orkestracija pomaže IT-u da lakše upravlja složenim sustavima, zadacima, radnim tijekovima i operacijama.

IT timovi moraju upravljati mnogim poslužiteljima, aplikacijama, sustavima … i „ručna“ administracija nije baš najbolja strategija. Što je IT sustav složeniji, upravljanje svim njegovim dijelovima može postati sve složenije i predstavljati sve veći izazov. Iz tih razloga, povećava se potreba za kombiniranjem više automatiziranih zadataka i njihovih konfiguracija u grupama sustava ili strojeva. Tu svakako može pomoći orkestracija.

Automatizacija i orkestracija su različiti, ali definitivno povezani koncepti.

Automatizacija pomaže učiniti poslovanje učinkovitijim smanjenjem ili zamjenom ljudske interakcije s IT sustavima korištenjem softvera za obavljanje zadataka kako bi se smanjili troškovi, složenost i što je možda i najvažnije, umanjila mogućnost pogreške.

Općenito, automatizacija se odnosi na automatizaciju jednog ili više zadatka prema jednoj ili više točaka. Ovo se razlikuje od orkestracije, što predstavlja način na koji možemo automatizirati proces/e ili tijek rada koji uključuju mnogo koraka na više različitih sustava.

Današnji IT sustavi donose složene implementacije i izazove. Moramo se nositi s klasteriranim aplikacijama, višestrukim i distribuiranim podatkovnim centrima, javnim, privatnim i hibridnim oblacima i aplikacijama sa složenim ovisnostima. Dakle, potreban je alat koji može jednostavno organizirati i osigurati da se svi ti zadaci odvijaju pravilnim redoslijedom, na vrijeme i po mogućnosti bez grešaka ili bolje rečeno sa što manje grešaka.

Orkestracija kontejnera (eng. Container Orchestration)

Orkestracija kontejnera može se koristiti u gotovo bilo kojem okruženju u kojem koristimo kontejnere. Može nam pomoći pri implementaciji iste aplikacije u različitim okruženjima bez potrebe za redizajniranjem. Mikroservisima u kontejnerima olakšava organizaciju usluga, uključujući pohranu, umrežavanje i sigurnost.

Orkestracija kontejnera automatizira implementaciju, upravljanje, skaliranje i umrežavanje kontejnera. Današnje tvrtke koje implementiraju i upravljaju velikim brojem kontejnera i čvorova na kojima se oni izvršavaju jednostavno su primorane koristiti sustave za orkestriranje kontejnera.

Kontejneri daju aplikacijama temeljenim na mikrouslugama idealnu jedinicu za implementaciju aplikacija (ili dijelova aplikacija) te samostalno, neovisno i izolirano okruženje za izvršavanje. Omogućuju neovisno pokretanje više dijelova aplikacije u mikroservisima, na istom hardveru, uz mnogo veću kontrolu nad pojedinačnim dijelovima i životnim ciklusima.

Upravljanje životnim ciklusom kontejnera s orkestracijom pomaže DevOps timovima koji ga onda mogu integrirati u CI/CD (eng. Continuous Integration/ Continuous Delivery) radni tijek.

Za što se sve koristi orkestracija kontejnera?

Orkestracija kontejnera koristi se za automatizaciju i upravljanje zadacima kao što su:

- Implementaciju kontejnera
- Konfiguracija i zakazivanje
- Alokacija resursa
- Dostupnost kontejnera
- Skaliranje (horizontalno) ili uklanjanje kontejnera
- Balansiranje opterećenja i usmjeravanje prometa
- Praćenje stanja kontejnera
- Konfiguriranje aplikacija na temelju kontejnera u kojem će se izvoditi
- Zaštita interakcije / komunikacije između kontejnera

Koji su najzastupljeniji alati za orkestraciju kontejnera?

Ovaj popis pokriva najzastupljenije alate za orkestraciju kontejnera koje mnoge organizacije u današnje vrijeme koriste kao produkcijsku okolinu za svoje sustave, aplikacije i sl. Popis se nalazi na slijedećem linku: https://devopscube.com/docker-container-clustering-tools/

Kontejnerska orkestracija u današnje vrijeme postala je de-facto standard. Danas su gotovo svi kontejneri orkestrirani, a sustave bazirane na sustavu Kubernetes (razne varijante i rješenja) koristi veliko broj organizacija.

2.1 Zašto nam je potreban sustav Kubernetes?

Ovo poglavlje podijelit ćemo u dva dijela i to:

- Zašto tehnološkim (IT) tvrtkama treba Kubernetes?
- Zašto je zajednici korisnika potreban Kubernetes?

I jedno i drugo je važno i oboje igraju važnu ulogu u tome zašto je Kubernetes ovdje i to po svemu sudeći, dugoročno. Neke od točaka također će nam pomoći da izbjegnemo potencijalne zamke kada započinjemo s projektom implementacije Kubernetes sustava što je isto tako vrlo važno za napomenuti.

Zašto tehnološkim tvrtkama treba Kubernetes ?

Sve počinje s AWS-om (eng. Amazon Web Services). Prije 2006. godine postojao je „status quo“ u tehnološkoj industriji. Velike tehnološke tvrtke lako su zarađivale prodajući poslužitelje, mrežne preklopnike, sustave za pohranu podataka, licence za razne operativne sustave, aplikacije i mnoge druge stvari.

Zatim, iz prikrajka Amazon je pokrenuo AWS i okrenuo svijet naglavačke. Bilo je to rođenje modernog računalstva u oblaku.

Isprva, nijedna od velikih tehnoloških tvrtki nije obraćala previše pozornosti tj. bile su previše zauzete zaradom prodajom istih stvari koje su prodavale desetljećima. Zapravo, neke od najvećih tehnoloških tvrtki mislile su da bi mogli okončati prijetnju AWS-a putem grubih kampanja dezinformacija. Mnoge od njih počeli su govoriti da oblak ne predstavlja budućnost razvoja IT industrije, općenito.

Kad to nije upalilo, „okrenuli su ploču“ i priznali da je oblak „OK“ i odmah „rebrendirali“ svoje postojeće naslijeđene proizvode u "oblak". Kada ni to nije upalilo, počeli su graditi vlastite oblake i usluge u oblaku i od tada se gotovo sve „vrti“ oko usluga u oblaku.

Kada je AWS počeo „krasti“ kupce i buduće poslovanje, industriji je trebao protunapad. Njihova prva velika osveta bila je OpenStack. Da skratimo priču, OpenStack je bio projekt zajednice koja je pokušala stvoriti alternativu otvorenog koda za AWS. Bio je to plemenit projekt i puno dobrih ljudi je tome doprinijelo.

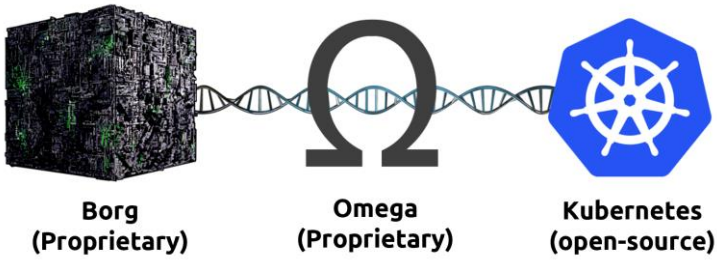
Ali, u konačnici to nikada nije ugrozilo AWS. Štoviše, Amazon je imao previše prednosti i razvijao se velikom brzinom. OpenStack se jako potrudio, ali AWS ga je odbacio bez oklijevanja.

Dok se sve to događalo, pa čak i prije, Google je, bez previše „buke“ uporno razvijao vlastiti sustav baziran na Linux kontejnerima za pokretanje većine svojih usluga.

Nije tajna da Google postavlja milijune kontejnera mjesečno otkad koristi tu tehnologiju. Planiranje i upravljanje tim milijunima kontejnera povjeren je internom alatu prvotno nazvanom „Borg“.

Google kao Google, tijekom vremena naučio je mnogo lekcija koristeći sustav „Borg“ i na toj platformi izgradio noviji sustav pod nazivom „Omega“.

U toj situaciji, neki ljudi unutar Googlea htjeli su iskoristiti lekcije naučene od Borga i Omege i izgraditi nešto bolje i učiniti to otvorenim kodom i dostupnim zajednici. I tako je nastao sustav Kubernetes u ljeto 2014. godine.



Slika 3. Razvoj sustava Kubernetes (Izvor: https://blog.risingstack.com/the-history-of-kubernetes/)

Dakle, Kubernetes nije verzija „Borg-a“ ili „Omege“ otvorenog koda. To je potpuno novi projekt, izgrađen od nule, sa ciljem da postane orkestrator kontejnerskih aplikacija otvorenog koda. Veza s „Borgom“ i „Omegom“ je jedno u tome da su njegovi početni programeri radili u

Google-u na tim sustavima, te da je izgrađen na temelju stečenih znanja i iskustava dobivenih iz tih vlastitih internih Google-ovih tehnologija.

No, vratimo se priči o AWS-u. Kada je Google predstavio Kubernetes otvorenog koda 2014. praktički u isto vrijeme, Docker je osvojio svijet. Kao rezultat toga, Kubernetes je prvenstveno viđen kao alat za pomoć korisnicima u upravljanju velikim rastom broja kontejnera.

"Apstraktna i komercijalna infrastruktura" je otmjeni način da se kaže da Kubernetes to čini tako da ne morate brinuti na čijem oblaku ili poslužiteljima rade vaše aplikacije. Ustvari, to je u središtu ideje da je Kubernetes zapravo operativni sustav (OS) u oblaku. Na isti način Linux i Windows znače da ne morate brinuti rade li vaše aplikacije na poslužiteljima Dell, Cisco ili HPE. Kubernetes znači da ne morate brinuti rade li vaše aplikacije na AWS, Azure, Google ili nekom drugom oblaku (javnom, privatnom, hibridnom ili nekom sasvim trećem).

Kubernetes je postao prilika za tehnološku industriju da izbriše vrijednost AWS-a, a to bi značilo da samo napišite svoje aplikacije koje će se izvoditi na Kubernetes sustavu i nije važno čiji je oblak ispod.

Zahvaljujući sustavu Kubernetes, teren je gotovo pa izjednačen. Ova sposobnost uklanjanja vrijednosti AWS-a glavni je razlog zašto dobavljači sustav Kubernetes stavljaju u središte svoje ponude jer velika većina najvažnijih davatelja usluga za svoje usluge u nazivu ima riječ „Kubernetes“ kao što su GKE, EKS, AKS i sl.. U svim slučajevima slovo „K“ predstavlja sustav Kubernetes.

To stvara snažnu, svijetlu i dugu budućnost za Kubernetes, što zauzvrat zajednici korisnika daje sigurnu platformu koja je neutralna prema dobavljačima i na koju se možemo kladiti u svoju budućnost u oblaku.

Zašto je zajednici korisnika potreban Kubernetes ?

Upravo smo naglasili dugu i svijetlu budućnost za sustav Kubernetes zajedno sa svim velikim tehnološkim tvrtkama koje ga podržavaju. Zapravo, tako je brzo rastao i postao toliko važan da ga je čak i Amazon nevoljko prihvatio. Tako čak ni moćni Amazon nije mogao zanemariti Kubernetes (koliko god se trudio) nudeći svoju „verziju“ preko EKS-a (eng. Elastic Kubernetes Services; https://aws.amazon.com/eks/) (!?).

U svakom slučaju, korisnička zajednica treba neovisne platforme na kojima se može graditi, sigurna da će te platforme biti i/ili postati dobra dugoročna ulaganja u tehnologiju.

Kako stvari stoje, izgleda da će Kubernetes biti s nama još jako dugo. Još jedan razlog zašto zajednica korisnika treba i zašto voli Kubernetes je povratak na pojam Kubernetes kao OS-a u oblaku. Već je rečeno da Kubernetes može apstrahirati „on-prem“ i infrastrukturu u oblaku na niže razine, omogućujući vam da napišete svoje aplikacije za rad u sustavu Kubernetes, a da uopće ne znate koji oblak stoji iza toga. U tom slučaju to nikako neće predstavljati problem.

Stoga, nakon svega izrečenog možemo zaključiti da:

- Danas aplikacije možemo implementirati u jedan oblak, a sutra se prebaciti na drugi,
- Aplikaciju/e, sustav … možemo pokrenuti multi-cloud okruženju,
- Lako se možemo „ukrcati“ na oblak, a zatim se vratiti u vlastiti podatkovni centar.

U osnovi, a što smo već u nekoliko navrata napomenuli (i to ne bez razloga!) aplikacije napisane za Kubernetes pokretat će se bilo gdje, gdje imamo postavljen Kubernetes klaster (ili neku njegovu verziju i/ili izvedenicu) bez obzira gdje se nalazimo. U oblacima ili na zemlji.

Sličnu analogiju možemo primijeniti i na primjeru pisanja aplikacija za operativni sustav Linux. Ako aplikacije pišemo za rad na Linuxu, nije važno radi li se o Linux sustavu na *Supermicro* poslužiteljima u obližnjoj garaži ili na AWS cloud instancama na drugoj strani planete.

Sve je to jako dobro i korisno za krajnje korisnike. U osnovi, pitamo se, tko ne bi poželio sustav/platformu koji nam donosi takovu fleksibilnost i ima vrlo solidnu i svijetlu budućnost?!

2.2 Varijante sustava Kubernetes

U orkestracijskom ekosustavu, Kubernetes je vodeći među svim varijacijama. Međutim, postoje i neki drugi koji igraju vrlo važnu ulogu. U nastavku ovog poglavlja predstaviti ćemo neke platforme kao varijante sustava Kubernetes, a koje služe različitim svrhama u cjelokupnom ekosustavu.

Ustvari, možemo reći da velika većina sustava za orkestraciju kontejnera bazirana su u svojoj srži na sustavu Kubernetes. Sve ostalo su „varijacije na temu“.

Cijeli CNCF „Cloud Native Interactive Landscape“ uključujući i kategoriju (Orchestration & Management) u kojoj se nalazi sustav Kubernetes nalazi se na slijedećem linku: https://landscape.cncf.io/ .

Slika 4. Cloud Native Interactive Landscape logo (Izvor: https://landscape.cncf.io/)

CNCF su 2015. godine osnovali *Google* i *Linux Foundation* kako bi pomogli unapređivanju tehnologija spremnika i uskladili tehnološku industriju oko evolucije ovog novog područja.

Kubernetes je prvi projekt koji se pridružio CNCF-u. Prvobitno kreiran od strane Googlea za internu upotrebu, Borg (izvorni naziv projekta) razvio je i koristio tehnologiju za orkestriranje spremnika. Kada je Google „otvorio projekt“, sustav Kubernetes prepisan je u „*Go*“ programski jezik (https://go.dev/), preimenovan u Kubernetes i doniran CNCF-u.

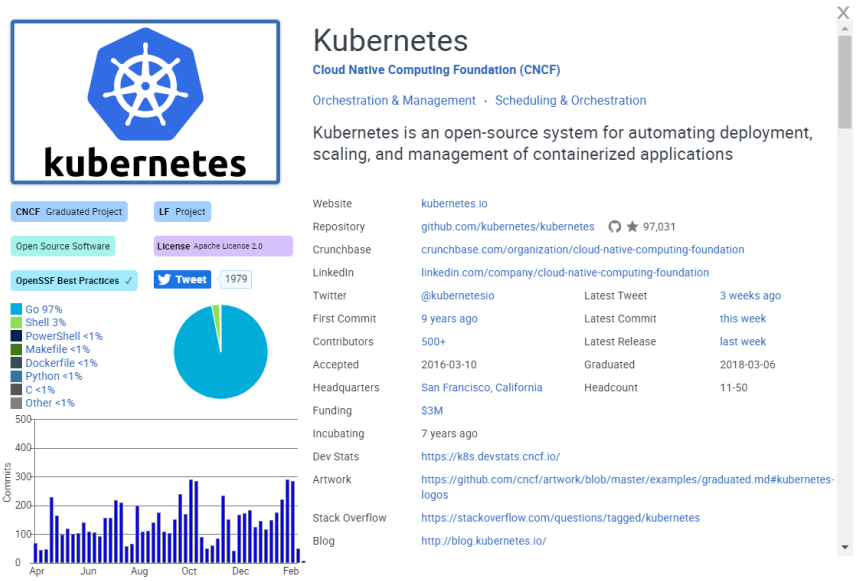
Kubernetes je bio samo početak. Ubrzo je uslijedio val tehnološkog napretka, a CNCF-u su se pridružili i mnogi drugi izvorni projekti u oblaku. Godine 2016. godine najavljen je program za uspostavu kvalificiranih predstavnika svih tehnologija kojima upravlja CNCF. Danas više od 450 organizacija članica podržava CNCF.

CNCF Landscape organizira tehnologije u skupine/kategorije, na temelju funkcionalnosti, kao što su raspoređivanje i orkestracija, baze podataka, registri spremnika i sl.

CNCF (kratica od „Cloud Native Interactive Landscape“) karta je resursa koja sadrži i kategorizira stotine projekata i alata izvornih u oblaku ili drugim riječima „cloud-native projects“. CNCF sadrži šest (6) osnovnih kategorija i to:

(1) **Provisioning** - uključuje alate potrebne za temeljnu infrastrukturu na kojoj će funkcionirati okruženja u oblaku i prateće tehnologije,
(2) **Runntime** - uključuje alate potrebne za pokretanje spremnika u izvornim okruženjima oblaka,
(3) **Orchestration & Management** - uključuje alate potrebne za orkestraciju i upravljanje spremnicima, aplikacijama i resursima kao što je i sustav Kubernetes,
(4) **App Definition & Development** - uključuje alate koji aplikacijama omogućuju slanje i pohranu podataka i alate potrebne za izradu i implementaciju aplikacija,
(5) **Observability & Analysis** - uključuje alate koji pomažu u praćenju aplikacija i upozoravaju dionike na bilo koji problem,
(6) **Platforms** - uključuje platforme koje spajaju višestruke funkcije i alate. Ove platforme pomažu u konfiguriranju i podešavanju višestrukih alata kako bi se organizacijama pomoglo da lakše usvoje tehnologije u oblaku.

Na **Slici 5.** nalaze se detalji vezani uz sustav Kubernetes koji se nalaze na CNCF web portalu i dostupni su na slijedećem linku: https://landscape.cncf.io/?category=orchestration-management&grouping=category&selected=kubernetes



Slika 5. Detalji o sustavu Kubernetes na CNCF (Izvor: https://landscape.cncf.io/)

MiniKube

Minikube je verzija sustava Kubernetes s jednim čvorom/hostom koja se može izvoditi na Linux, Mac OS i Windows platformama.

Minikube podržava standardne značajke sustava Kubernetes, kao što su *LoadBalancer,* usluge, *PersistentVolume*, *Ingress*, kontejner *runntime* i značajke prilagođene razvojnim programerima kao što su npr. dodaci i podrška za GPU i sl.

Minikube je odlično početno mjesto za učenje, stjecanje znanja i praktičnog iskustva s sustavom Kubernetes. To je također dobro mjesto za lokalno pokretanje testova (na osobnim računalima), ili rad na dokazima koncepata (eng. PoC; Proof of Concept). Detaljnije o Minikube može se naći na slijedećem linku: https://minikube.sigs.k8s.io/docs/

K3s (eng. Lightweight Kubernetes)

K3s je „lagana“ Kubernetes platforma. Njegova ukupna veličina je manja od 40 MB. Izvrstan je za Edge, Internet stvari (eng. Internet-of-Things) i ARM. Skoro pa u potpunosti je usklađen s sustavom Kubernetes. Jedna značajna razlika jest u tome što on koristi SQLlite bazu podataka kao mehanizam za pohranu podataka, dok sustav Kubernetes koristi ***etcd*** kao poslužitelj za pohranu. Detaljnije o K3s može se naći na slijedećem linku: https://www.k3s.io/

Red Hat „OpenShift“

OpenShift usvojila je Docker kao svoju tehnologiju kontejnera i Kubernetes kao svoju tehnologiju orkestracije kontejnera. U verziji 4, OpenShift se prebacio na CRI-O kao zadano vrijeme izvođenja kontejnera (eng. Container Runtime). Na prvi pogled, čini se da bi OpenShift trebao biti isti kao Kubernetes; međutim, postoje određene (neke od njih i značajne) razlike.

Isto tako sustav OpenShift možemo pokrenuti na dva osnovna načina i to:

1. Kao „Cloud services“ edicije na slijedećim pružateljima usluga u oblaku:

- Microsoft Azure Red Hat OpenShift
- Red Hat OpenShift Dedicated
- Red Hat OpenShift on IBM Cloud
- Red Hat OpenShift Service on AWS

2. Kao „Self-managed“ edicije:

- Red Hat OpenShift Platform Plus
- Red Hat OpenShift Container Platform
- Red Hat OpenShift Kubernetes Engine

Slika 6. Red Hat OpenShift KE logo
(Izvor: https://www.redhat.com/en/technologies/cloud-computing/openshift)

Terminologija

Objekti imenovani u sustavu Kubernetes mogu imati različita imena u OpenShiftu, iako je ponekad njihova funkcionalnost vrlo slična. Na primjer, „*Namespace“* u sustavu Kubernetes naziva se „*Project*“ u OpenShift-u. „*Ingress“* u sustavu Kubernetes naziva se „*Routes“* u OpenShift-u.

Rute OpenShift implementira kao *HAProxy*, dok u sustavu Kubernetes postoji opcija kontrolera ulaza tj. „*Ingress“* koja kontrolira ulazni promet prema K8s klasteru tj. prema određenoj aplikaciji.

Sigurnost

Kubernetes je prema zadanim postavkama otvoren i manje tj. ne toliko siguran. OpenShift je relativno zatvoren sustav i nudi podosta dobrih sigurnosnih mehanizama za osiguranje klastera već prilikom instalacije tj. prema zadanim postavkama sustava.

Na primjer, pri izradi OpenShift klastera, može omogućiti postavljanje internog registra Image-a koji nije izložen vanjskom svijetu. U isto vrijeme, interni registar Image-a služi i kao pouzdani registar gdje će se Image povući i/ili rasporediti. Isti se onda može integrirati sa nekim sustavom za skeniranje Imagea na ranjivosti što dodatno doprinosi sigurnosti kako klastera tako i aplikacija, kontejnera i sl.

Trošak

OpenShift je proizvod koji nudi Red Hat (https://www.redhat.com/en/technologies/cloud-computing/openshift), iako postoji projekt tj. OSS verzija pod nazivom OpenShift Origin.

Kada se govori o OpenShift-u kao produkcijskom okruženju, obično se misli na opciju koja se plaća uz podršku Red Hat-a. Za razliku od te činjenice, sustav Kubernetes je potpuno besplatan sustav otvorenog koda!

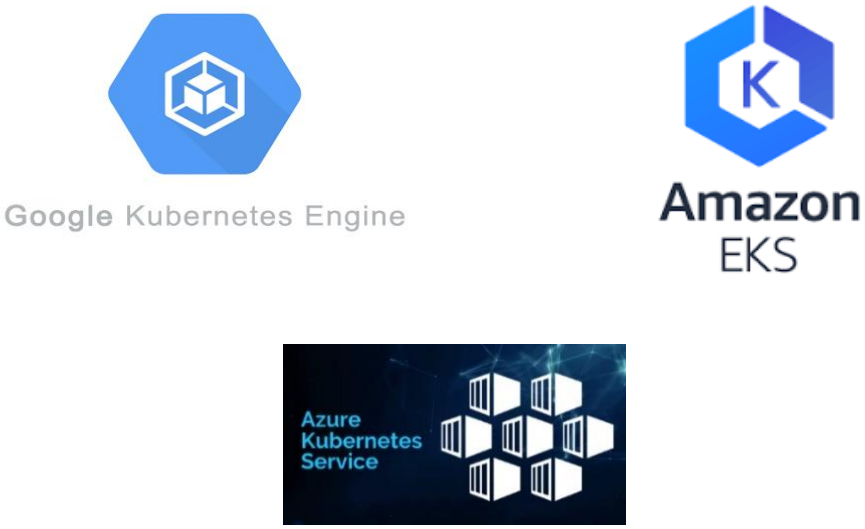
Kubernetes kao usluga u oblaku (eng. Kubernetes-as-a-Service; KaaS)

Kontejnerizacija čini aplikacije prenosivim tako da „zaključavanje sustava“ kod nekog određenog davatelja usluge u oblaku postaje vrlo malo vjerojatna.

Iako postoje neki sjajni alati otvorenog koda, kao što su *KubeAdm* i *Kops*, koji mogu pomoći u stvaranju Kubernetes klastera, Kubernetes kao usluga koje nude pružatelji usluga u oblaku i dalje zvuči privlačno. No, za tu varijantu u većini slučajeva potrebno je odvojiti poprilično novca.

Kao izvorni kreator sustava Kubernetes, Google nudi Kubernetes kao uslugu od 2014. i zove se Google Kubernetes Engine (GKE). Microsoft je 2017. ponudio vlastitu uslugu sustava Kubernetes pod nazivom Azure Kubernetes Service (AKS). AWS je ponudio Elastic Kubernetes uslugu (EKS) tek 2018.

Na slijedećoj poveznici nalazi se popis najpoznatijih pružatelja usluga u oblaku koji nude Kubernetes kao KaaS (eng. Kubernetes-as-a-Service): https://www.aquasec.com/cloud-native-academy/kubernetes-101/kubernetes-as-a-service/

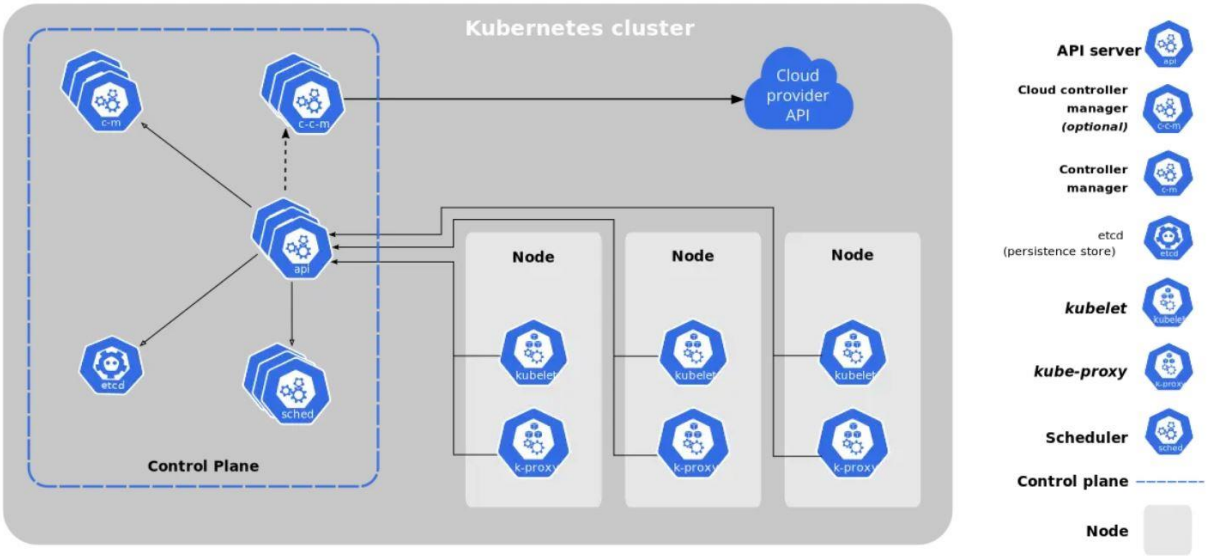


Slika 7. Neki od pružatelja usluga KaaS u oblaku (Izvor: https://aws.amazon.com/eks/, https://azure.microsoft.com/en-us/services/kubernetes-service/, https://cloud.google.com/kubernetes-engine)

2.3 Način rada, arhitektura i komponente sustava Kubernetes

U ovom poglavlju obraditi će se glavne komponente potrebne za izgradnju Kubernetes klastera i implementaciju aplikacija. Cilj je dati osnovni pregled koncepta sustava i ne ulaziti u detalje. Ovo poglavlje, između ostaloga opisuje slijedeće dijelove, komponente i objekte sustava Kubernetes:

- Kubernetes - najviša razina/pogled
- Glavni (Master) čvorovi/node-ovi - upravljačka razina
- Radni (Working) čvorovi/node-ovi - radna/izvršna razina
- Pakiranje aplikacija/mikroservisa za Kubernetes
- Deklarativni model i željeno stanje
- Pod
- Ingress kontroler
- Deployment
- Servisi



Slika 8. Osnovna arhitektura sustava Kubernetes
(Izvor: https://kubernetes.io/docs/concepts/overview/components/)

Na najvišoj razini, sustav Kubernetes podrazumijeva dva dijela:

- Klaster za pokretanje aplikacija
- Orkestraciju aplikacija tj. mikroservisa unutar klastera

Kubernetes kao klaster

Kubernetes čini, kao i svaki drugi klaster, određeni broj računala (servera ili strojeva) za udomljavanje (eng. Hosting) aplikacija povezanih u jednu logičku cjelinu. Te strojeve, servere ili računala nazivamo "čvorovi" (eng. Nodes), a oni mogu biti fizički poslužitelji, virtualni strojevi, instance u oblaku, Raspberry Pi-s, ili naprosto „obična“ stolna računala.

Kubernetes klaster se sastoji od kontrolne razine (eng. Master Node) i radnih čvorova (eng. Working Node). Upravljačka razina izlaže API, ima planer za dodjelu „posla“ i bilježi stanje klastera (i aplikacija) u svojoj internoj „*etcd*“ bazi podataka.

Radni čvorovi su mjesto gdje se pokreću korisničke aplikacije. Može biti korisno razmišljati o kontrolnoj razini kao o „mozgu“ Kubernetes klastera, a o čvorovima kao o „mišićima“ koji odrađuju određeni konkretan posao.

U ovoj analogiji, upravljačka razina implementira pametne značajke kao što su zakazivanje (eng. Scheduling), automatsko skaliranje i sl. dok radni čvorovi obavljaju svakodnevni naporan posao izvršavanja korisničkih aplikacija.

Kubernetes kao orkestrator

Orkestrator je samo pojam za sustav koji se brine za implementaciju i upravljanje aplikacijama tj. „cloud-native“ rječnikom rečeno, mikroservisima.

Neki servisi/app poslužuju web stranice, neki provjeravaju autentičnost, neki pretražuju, drugi pohranjuju podatke. U takovoj situaciji do izražaja dolazi Kubernetes. Poput trenera u sportskoj analogiji on organizira sve u korisnu cjelinu i održava sustav bez problema. Čak reagira i na događaje sa druge razine kao što su kvarovi čvorova i problemi s mrežom. U sportskom svijetu to se zove „coaching“. U svijetu aplikacija to se zove orkestracija. Kubernetes orkestrira aplikacije, mikroservise, sam klaster i njegove sastavne komponente.

Kako radi sustav Kubernetes

Počinjemo s aplikacijom, „pakiramo“ ju kao kontejner, a zatim predajemo klasteru tj. sustavu Kubernetes. Klaster se sastoji od jednog ili više čvorova (eng. Nodes), upravljačke razine i većeg broja radnih čvorova.

Kao što svo već napomenuli, čvorovi kontrolne razine implementiraju inteligenciju klastera, a radni čvorovi su mjesto gdje se pokreću i rade korisničke aplikacije.

Slijedi pojednostavljen postupak za pokretanje aplikacija u Kubernetes klasteru i to redom:

1. Dizajn aplikacije kao skupa neovisnih mikroservisa
2. Pakiranje mikroservisa u vlastite kontejnere
3. Pakiranje svakog kontejnera u Kubernetes Pod
4. Postavljanje Pod-ova u klaster putem kontrolera više razine

Na višoj razini, Kubernetes ima nekoliko kontrolera koji opslužuju Pod-ove i kontejnere važnim značajkama kao što su samoizlječenje, skaliranje, uvođenje i još mnogo toga. Neki kontroleri su zaduženi za aplikacije bez stanja (eng. Stateless apps), a drugi za aplikacije s statusom (eng. Stateful apps).

Kubernetes deklarativno upravlja aplikacijama. Ovo je obrazac u kojem se opisuje što želite u skupu konfiguracijskih datoteka, postavljate ih na Kubernetes, a Kubernetes zatim učini sve što mu je rečeno. Ali ne staje tu. Budući da deklarativni model govori sustavu Kubernetes kako bi aplikacija trebala izgledati, Kubernetes je može gledati i osigurati da se stanje ne razlikuje od onoga što ste tražili tj. što mu je rečeno i zapisano u konfiguracijskoj datoteci. Ako nešto nije kako bi trebalo biti, Kubernetes to pokušava popraviti.

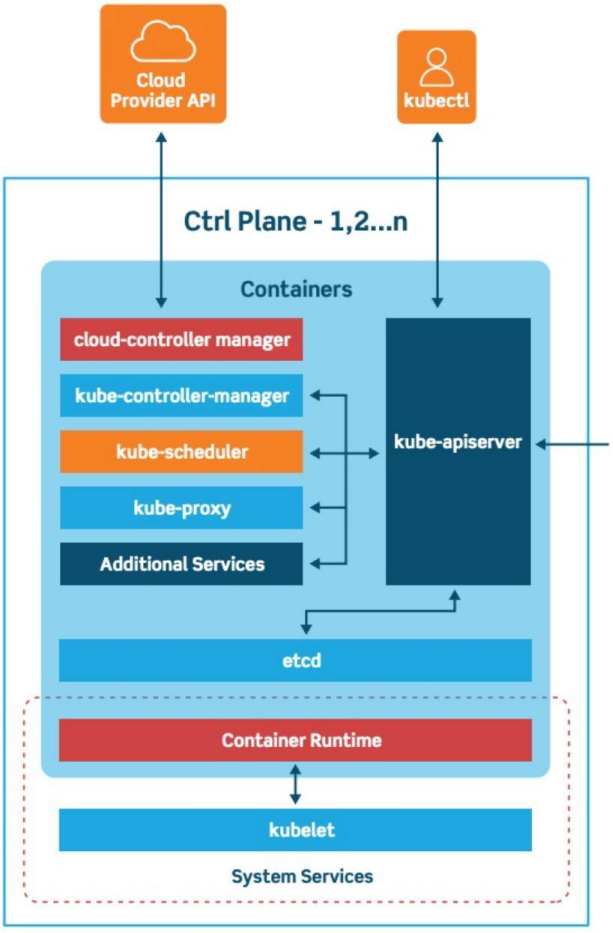
Kontrolna razina - glavni (upravljački) čvorovi (eng. Control Plane; Master Node)

Kao što je već spomenuto, Kubernetes klaster sastoji se od čvorova upravljačke razine i radnih čvorova. To su ustvari Linux hostovi koji mogu biti virtualni strojevi (VM), fizički serveri u podatkovnim centrima ili instance u privatnom ili javnom oblaku.

Čvor Kubernetes kontrolne razine (eng. Master Node) pokreće kolekciju sistemskih usluga koje čine kontrolnu razinu klastera. Ponekad ih nazivamo Master, Heads ili Head nodes. Osnovne postavke pokreću jedan čvor upravljačke razine.

Međutim, ovo je prikladno samo za laboratorijska i testna okruženja. Za produkcijska okruženja od vitalnog je značaja postaviti više čvorova upravljačke razine konfiguriranih za visoku dostupnost (eng HA; High Availability). Općenito govoreći, preporučuje se 3 ili 5 čvorova upravljačke razine, a potrebno ih je rasporediti po zonama dostupnosti (eng. Availability Zones).

Također, smatra se dobrom praksom ne pokretati korisničke aplikacije na čvorovima upravljačke razine. To im omogućuje da se u potpunosti usredotoče na upravljanje klasterom tj. na upravljačke funkcije.



Slika 9. Čvor kontrolne razine (eng. Master Node); Kontrolna razina (eng. Control Plane) sustava Kubernetes (Izvor: https://platform9.com/blog/kubernetes-enterprise-chapter-2-kubernetes-architecture-concepts/)

API poslužitelj

API (eng. Application Programming Interface) poslužitelj je „centralna stanica“ sustava Kubernetes. Sva komunikacija, između svih komponenti, mora ići kroz API poslužitelj. Vrlo je važno razumjeti da unutarnje komponente sustava, kao i vanjske korisničke komponente, sve komuniciraju putem API poslužitelja. Rekli bi smo, svi putevi vode do API poslužitelja.

API poslužitelj izlaže API kroz koji postavljamo YAML konfiguracijske datoteke preko HTTPS-a. Ove YAML datoteke, koje ponekad nazivamo manifestima, opisuju željeno stanje aplikacije.

Ovo željeno stanje uključuje stvari poput toga koju sliku (eng. Image) kontejnera koristiti, koje portove izložiti te koliko replika Pod-a želimo. Svi zahtjevi prema API poslužitelju podliježu provjerama autentičnosti i autorizaciji. Kada se to odradi, konfiguracija u YAML datoteci provjerava valjanost, ostaje u spremištu klastera (**etcd** bazi podataka) i rad se raspoređuje po radnim čvorovima ovisno gdje se aplikacija tj. pripadajući kontejner nalazi.

Pohrana klastera

Pohrana klastera jedini je dio kontrolne razine sa stanjem koja trajno pohranjuje cijelu konfiguraciju i stanje klastera. Kao takva, to je vitalna komponenta svakog Kubernetes klastera - nema pohrane klastera, nema klastera.

Pohrana klastera trenutno se temelji na ***etcd,*** popularnoj distribuiranoj bazi podataka. Etcd možemo opisati kao distribuirano, pouzdano spremište za najkritičnije podatke distribuiranog sustava.

Budući da je to jedini izvor informacija za klaster, potrebno je pokrenuti između 3-5 *etcd* replika za visoku dostupnost. Također, potrebno je osigurati odgovarajuće načine za oporavak kada stvari pođu po zlu. Zadana instalacija sustava Kubernetes instalira repliku spremišta klastera na svaki čvor kontrolne razine i automatski konfigurira HA.

Na temu dostupnosti, *etcd* preferira dosljednost nad dostupnošću. To znači da ne tolerira podijeljene mozgove i da će zaustaviti ažuriranja klastera kako bi održala dosljednost.

Međutim, ako se to dogodi, korisničke aplikacije trebale bi nastaviti raditi i u tom trenutku jednostavno neće biti moguće moći ažurirati konfiguraciju klastera. Kao i kod svih distribuiranih baza podataka, dosljednost upisivanja u bazu podataka je od vitalnog značaja.

Na primjer, potrebno je obraditi višestruka upisivanja iste vrijednosti koja potječu s različitih mjesta. *Etcd* koristi popularni RAFT konsenzus algoritam da to postigne (ne ulazeći u dubinu pojašnjenja RAFT algoritma jer to nadilazi temu ovog specijalističkog rada).

Upravitelj kontrolera (eng. Kube Controller Manager) **i kontroleri** (eng. Controllers)

Upravitelj kontrolera implementira sve pozadinske kontrolere koji nadziru komponente klastera i reagiraju na određene događaje. Prema arhitekturi sustava Kubernetes, upravitelj kontrolera je kontroler kontrolera, što znači da stvara sve „sebi podređene" kontrolere i nadzire ih.

Neki od osnovnih kontrolera uključuju kontroler za implementaciju, kontroler *StatefulSet* i kontroler *ReplicaSet.* Svaki od njih je odgovoran za mali podskup inteligencije klastera i radi kao pozadinska petlja za praćenje koja neprestano promatra API poslužitelj za promjene. Cilj svakog kontrolera je osigurati da promatrano stanje klastera odgovara željenom stanju.

Logika koju implementira svaki kontroler je sljedeća i u središtu je sustava Kubernetes i deklarativnih obrazaca dizajna:

1. Dobiti željeno stanje
2. Promatrajte trenutno stanje
3. Odredite razlike
4. Pomirite razlike

Svaki kontroler je također iznimno specijaliziran i zainteresiran samo za svoj mali kutak Kubernetes klastera. Svaki se kontrolor brine o svom poslu, a sve ostalo ostavlja na miru. To je ključno za distribuirani dizajn sustava Kubernetes i pridržava se Unix filozofije izgradnje složenih sustava od malih specijaliziranih dijelova.

Upravitelj kontrolera oblaka (eng. Cloud Controller Manager)

Ako se pokreće klaster na platformi u oblaku, kao što je GKE, AWS, Azure ili Linode, kontrolna razina pokretati će upravitelj kontrolera oblaka. Njegov je posao olakšati integraciju s uslugama u oblaku, kao što su instance, balanseri i pohrana (eng Storage). Na primjer, ako aplikacija traži balanser opterećenja (eng. Load Balancer) okrenut internetu, upravitelj kontrolora oblaka radi s temeljnim oblakom kako bi osigurao *Load Balancer* i povezao ga s dediciranom aplikacijom.

Radni čvorovi (eng. Worker Nodes)

Radni čvorovi su mjesto gdje se postavljaju i pokreću korisničke aplikacije. Na visokoj razini rade tri stvari:

1. Prate i/ili „slušaju“ API poslužitelj za nove radne zadatke
2. Izvršavaju radne zadatke
3. Šalju povratne poruke prema kontrolnoj razini (putem API poslužitelja)

Kubelet

Kubelet je glavni Kubernetes agent i radi na svakom radnom čvoru. Kada čvor pridružimo klasteru, proces instalira *Kubelet*, koji je zatim odgovoran za njegovu registraciju u klasteru. Ovaj proces registrira CPU, memoriju i pohranu čvora u širi skup klastera.

Jedan od glavnih zadataka *Kubelet-a* je nadgledanje API poslužitelja za nove radne zadatke. Svaki put kad ga vidi, izvršava zadatak i održava kanal za izvješćivanje natrag na kontrolnu razinu.

Ako *Kubelet* ne može pokrenuti zadatak, javlja se kontrolnoj razini i dopušta joj da odluči koje radnje poduzeti. Na primjer, ako *Kubelet* ne može izvršiti zadatak, nije odgovoran za pronalaženje drugog čvora na kojem će ga pokrenuti. Jednostavno se javlja kontrolnoj razini i kontrolna razina odlučuje što će učiniti.

Vrijeme izvođenja kontejnera (eng. Container Runtime)

Kubelet-u je potrebno vrijeme izvođenja (eng. Runtime) kontejnera za obavljanje zadataka povezanih s kontejnerom tj. detalji kao što su povlačenja slika (eng. Images) te pokretanja i zaustavljanja kontejnera.

U ranim danima, Kubernetes je imao izvornu podršku za Docker Engine/Runtime to sve do verzije 1.20. Novije verzije Kubernetes (nakon 1.20) koriste Container Runtime Interface (CRI). No, to ne znači da Docker neće raditi u sustavu Kubernetes. Dapače, još uvijek veliki broj sustava baziran na sustavu Kubernetes koristi Docker Runtime.

Kube Proxy

Posljednji dio slagalice radnog čvora je *kube-proxy*. On radi na svakom čvoru i odgovoran je za lokalno umrežavanje klastera tj. odrađuje sve one poslove vezane uz mrežu. Osigurava da svaki čvor dobije svoju jedinstvenu IP adresu i implementira lokalne *iptables* (ili *IPVS*) pravila za rukovanje usmjeravanjem i balansiranjem opterećenja prometa na Pod mreži.

Kubernetes DNS

Svaki Kubernetes klaster ima interni DNS koji je od vitalnog značaja za cjelokupan sustav. DNS usluga klastera ima statičku IP adresu koja je zapisana u svaki Pod na klasteru. To osigurava da ga svaki kontejner i Pod mogu locirati i koristiti za otkrivanje (eng. Discovery). Registracija usluge je također automatska. To znači da aplikacije ne moraju biti kodirane s inteligencijom da bi se registrirale na *Kubernetes service discovery*. Cluster DNS temelji se na *CoreDNS* projektu otvorenog koda. Više detalja može se naći na: https://coredns.io/.

Priprema aplikacija za sustav Kubernetes (eng. Packaging Apps)

Aplikacija treba označiti nekoliko okvira da bi se pokrenula na Kubernetes klasteru. To uključuje slijedeće:

1. App pripremljena kao kontejner
2. „Zapakirana“ u Pod
3. Razmještena (eng. Deployed) putem manifest datoteke

Cijeli postupak ide otprilike ovako:

Mikroservis aplikacije napisan je u nekom od programskih jezika. Zatim ga se ugrađuje sliku (eng. Image) kontejnera i pohranjuje u registar.

Zatim, definiramo Kubernetes Pod za pokretanje kontejnerizirane aplikacije/mikroservisa. Na visokoj razini na kojoj se nalazimo, Pod je samo omot koji omogućuje da kontejner radi u Kubernetes klasteru. Nakon što se definira pripadajući Pod, sve je spremno za implementaciju aplikacije ili mikroservisa kao dijela aplikacije u sustavu Kubernetes.

Iako je moguće pokrenuti statične Pod-ove, preferirani model je implementirati sve Pod-ove putem kontrolera više razine. Najčešći kontroler je *Deployment*. Nudi skalabilnost, samoizlječenje i ažuriranja za aplikacije. *Deployment* se definira u YAML manifest datotekama koje određuju stvari kao što su npr. koliko replika treba implementirati, koji *Image* koristiti i sl.

Nakon što je sve definirano u YAML datoteci za implementaciju, možemo koristiti Kubernetes komandni liniju (eng. Command-line-interface; CLI) ili ***kubectl*** kako bi ga objavili API poslužitelju kao željeno stanje aplikacije, a Kubernetes će ga zatim implementirati.

Primjer jednostavne YAML *Deployment* datoteke za *Busybox*:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: primjer-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: primjer-deployment-app
  template:
    metadata:
      labels:
        app: primjer-deployment-app
    spec:
      containers:
        - name: busybox
          image: busybox
          command:
            - sleep
            - "3600"
```

Deklarativni model (eng. Declarative Model) **i „željeno stanje“** (eng. Desired State)

Deklarativni model i koncept „željenog stanja“ u samom su srcu sustava Kubernetes. Stoga je jako važno da ih se u potpunosti razumije. U sustavu Kubernetes deklarativni model funkcionira ovako:

1. Deklariramo željeno stanje mikroservisa aplikacije u manifest datoteci
2. Objavimo ga na API poslužitelju
3. Kubernetes ga pohranjuje u pohranu klastera kao željeno stanje aplikacije
4. Kubernetes implementira željeno stanje na klaster
5. Kontroler osigurava da se trenutno stanje aplikacije ne razlikuje od željenog stanja

Manifest datoteke su napisane u YAML programskom jeziku i govore sustavu Kubernetes kako bi aplikacija trebala izgledati. To se zove „željeno stanje“. Uključuje detalje kao što su koju sliku koristiti (eng. Container Image), koliko replika pokrenuti, na kojim mrežnim priključcima (eng. Ports) slušati i kako izvršiti ažuriranja.

Nakon što se izrade manifest datoteke, objavljuju se na API poslužitelju. Jednostavan način da se to učini jest pomoću ***kubectl*** uslužnog programa naredbenog retka ili komandne linije. *Kubectl* šalje manifest na API poslužitelj kao HTTP POST, obično na portu 443.

Nakon što je zahtjev autentificiran i autoriziran, Kubernetes pregleda manifest, identificira kojem kontroloru da ga pošalje (npr., kontroloru za implementacije) i bilježi konfiguraciju u spremištu klastera kao dio ukupnog željenog stanja. Kada se to učini, svi potrebni radni zadaci se raspoređuju na radne čvorove gdje *kubelet* koordinira rad povlačenja slika, pokretanja kontejnera, spajanja na mrežu i konačno pokretanja procesa aplikacije.

U konačnici, kontroleri rade tako da neprestano prate stanje klastera. Ako se promatrano stanje razlikuje od željenog stanja, Kubernetes izvršava zadatke potrebne za rješavanje problema. Tu leži ključ cijelog procesa.

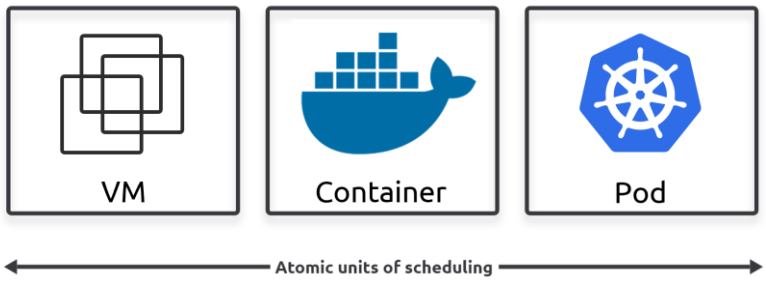
Ne samo da je deklarativni model puno jednostavniji od dugih skripti s puno imperativnih naredbi, on također omogućuje „samoizlječenje“, skaliranje i podložan je kontroli verzija.

Sve to čini tako što *Deployment* klasteru govori kako bi stvari trebale izgledati. Ako počnu izgledati drugačije, odgovarajući kontroler uočava neusklađenost i razlike te poduzima sav težak posao da pomiri situaciju tj. da uskladi trenutno sa željenim stanjem.

Pod

U svijetu VMware-a, HyperV-a osnovna jedinica je virtualni stroj (VM). U Docker svijetu to je kontejner. U svijetu Kubernetesa, to je ***Pod***. Istina jest da sustav Kubernetes u suštini pokreće kontejnerske aplikacije, međutim, Kubernetes zahtijeva da svaki kontejner radi unutar *Pod-a.*

Pod-ovi su objekti sustava Kubernetes, stoga prvo slovo pišemo velikim slovom. To daje jasnoću i službeni Kubernetes dokumenti također koriste taj standard. Tako je i sa ostalim glavnim komponentama Kubernetes kao što su *Service, Ingrss, Deployments* i sl.



Slika 10. Pod - osnovna jedinica sustava Kubernetes
(Izvor: https://chandanbhagat.com.np/understanding-pods-in-kubernetes/)

Primjer jednostavne Pod YAML datoteke:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-primjer-busybox
spec:
  containers:
    - command:
        - sleep
        - "3600"
      image: busybox
      name: container-primjer-busybox
```

Pod i kontejner

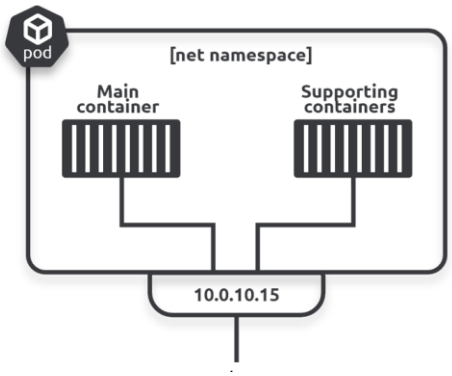
Prva stvar koju treba razumjeti je da izraz *Pod* dolazi od „Pod of Whales“ - na engleskom jeziku grupu kitova nazivamo „pod of whales“. Kako je Docker-ov logo kit, Kubernetes je izašao s konceptom kita i zato danas imamo "*Pod* ili *Pods*".

Najjednostavniji (a u većini slučajeva i najbolji) model je pokrenuti jedan kontejner po svakom *Pod-u.* Zbog toga često koristimo izraze "Pod" i "kontejner" naizmjenično. Međutim, postoje napredni slučajevi upotrebe koji pokreću više kontejnera unutar jednog Pod-a. Poanta je u tome da je Kubernetes *Pod* konstrukcija za pokretanje jednog ili više kontejnera.

Pod anatomija

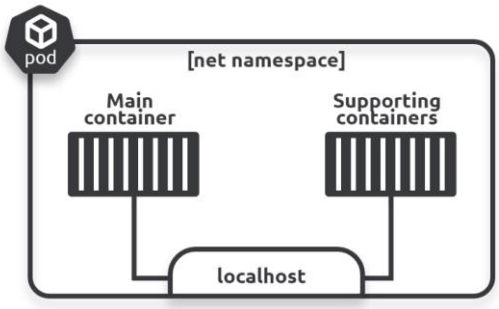
Na najvišoj razini, *Pod* je ograđeno okruženje za pokretanje kontejnera. Sami Pod-ovi zapravo ne pokreću aplikacije – aplikacije se uvijek pokreću u kontejnerima, *Pod* je samo „sandbox“ za pokretanje jednog ili više kontejnera. Održavajući ga na visokoj razini, *Pod* ograđuje područje glavnog OS-a, gradi mrežni sloj te pokreće jedan ili više kontejnera.

Ako pokrećemo više kontejnera u Pod-u, svi dijele isto Pod okruženje. To uključuje mrežni stog, volumene, imenski prostor (eng. Namespace), zajedničku memoriju i još mnogo toga. Na primjer, to znači da će svi spremnici u istom Podu dijeliti i istu IP adresu (IP Poda).



Slika 11. Varijanta 1 - kontejneri na istoj IP adresi unutar jednog Pod-a i Namespace-a
(Izvor: https://chandanbhagat.com.np/understanding-pods-in-kubernetes/)

Ako dva kontejnera u istom Podu trebaju međusobno „razgovarati“ (kontejner-kontejner unutar Pod-a), mogu koristiti lokalno sučelje Pod-a.



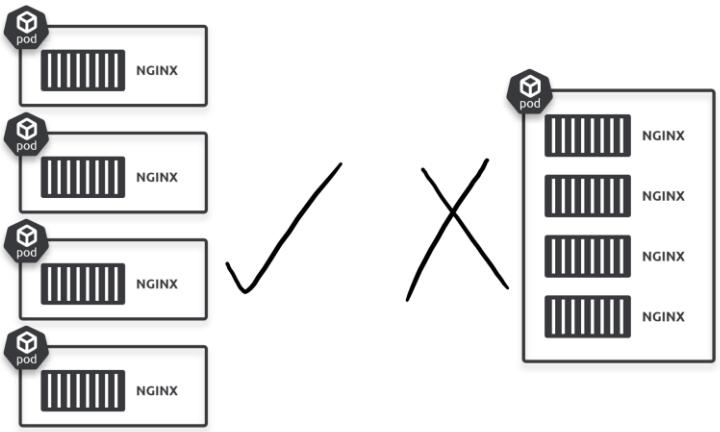
Slika 12. Varijanta 2 - dva kontejnera unutar istog Pod-a pod lokalnim sučeljem Pod-a
(Izvor: https://chandanbhagat.com.np/understanding-pods-in-kubernetes/)

Pod-ovi s više kontejnera idealni su kada imamo zahtjeve za čvrsto spojenim kontejnerima koji možda trebaju dijeliti memoriju i pohranu. Međutim, ako ne trebamo čvrsto spajati kontejnere, uputno je staviti ih u njihove vlastite Pod-ove i spojiti ih preko interne mreže.

To održava stvari čistima jer je svaki Pod posvećen jednom zadatku. Međutim, stvara mnogo potencijalno nešifriranog mrežnog prometa. U tom slučaju treba ozbiljno razmisliti o korištenju mreže na način da se osigurava sigurnost prometa između Podova i bolja vidljivost mreže.

Pod kao jedinica skaliranja

Pod-ovi su također minimalna jedinica u sustavu Kubernetes. Ako je potrebno skalirati aplikaciju, dodajemo ili uklanjamo Pod-ove. To se zove i **horizontalno skaliranje.** Skaliranje ne odrađujemo na način dodavanjem više kontejnera postojećim Pod-ovima. Pod-ovi s više kontejnera su samo i isključivo za situacije u kojima dva različita kontejnera trebaju/moraju dijeliti određene resurse ili moraju iz nekog vrlo određenog razloga međusobno „razgovarati“ na taj način..



Slika 13. Preporučeni način postavljanja Pod-ova (1 Pod = 1 kontejner)
(Izvor: https://chandanbhagat.com.np/understanding-pods-in-kubernetes/)

Životni ciklus Pod-ova

Pod-ovi su „smrtni“. Oni se stvaraju, žive i umiru. Ako neočekivano umru, ne vraćamo ih u život. Umjesto toga, Kubernetes pokreće novi Pod umjesto njega. Međutim, iako novi Pod izgleda kao stari, on to ustvari nije. To je novi Pod koji, doduše radi iste zadatke kao i stari, ali sa novim ID-em i IP adresom! Te činjenice su vrlo važne za zapamtiti!

To ima implikacije na način na koji dizajniramo aplikacije. Aplikacije se ne dizajniraju tako da budu čvrsto povezane s određenim Podom. Umjesto toga, dizajniraju se tako da kada Pod „umre“, potpuno novi (s novim ID-om i IP adresom) može „iskočiti“ negdje drugdje u klasteru (npr. na nekom drugom radnom čvoru) i neprimjetno zauzeti njegovo mjesto.

Pod nepromjenjivost

Pod-ovi su također nepromjenjivi - to znači da ih ne mijenjamo nakon što su pokrenuti. Na primjer, kada se Pod pokrene, nikada se ne prijavljujemo na njega i mijenjamo ili ažuriramo njegovu konfiguraciju.

Ako se mora promijeniti ili ažurirati Pod, zamjenjuje ga se novim koji pokreće novu konfiguraciju. Kad god govorimo o ažuriranju Podova, zapravo podrazumijevamo na brisanje starog i zamjenu novim Pod-om. To je isto tako vrlo važna činjenica i primjer dobre prakse unutar sustava Kubernetes.

Deployments

U većini situacija postavljat ćemo Pod-ove neizravno putem kontrolera više razine. Primjeri kontrolera više razine uključuju *Deployments, DaemonSets i StatefulSets.*

Na primjer, *Deployment* je Kubernetes objekt više razine koji se omota oko Poda i dodaje značajke kao što su samoizlječenje, skaliranje, uvođenje bez zastoja i verzionirana vraćanja u prijašnje stanje (eng. RollBack).

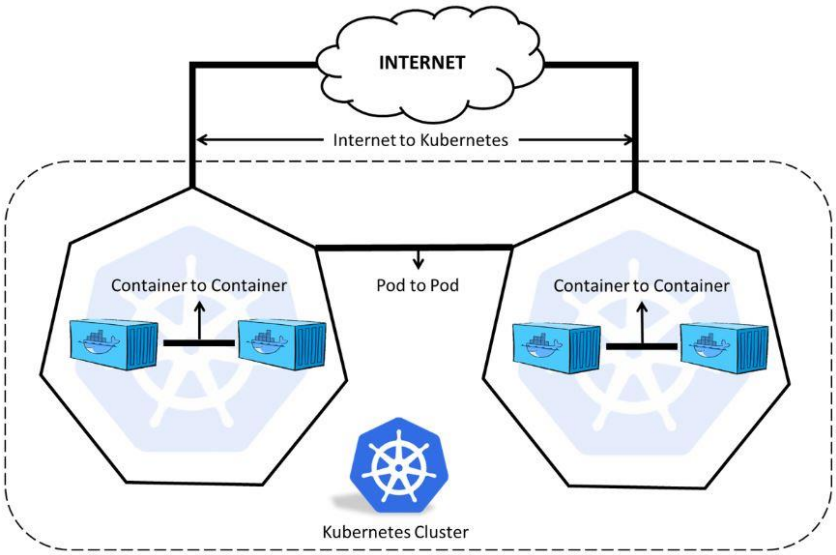
Iza kulisa implementirani su *Deployments, DaemonSets i StatefulSets* kao kontroleri koji se pokreću kao petlje koje neprestano promatraju klaster pazeći da promatrano stanje odgovara željenom stanju.

Servisni objekti i mrežni model

Upravo je spomenuto da su Pod-ovi „smrtni“ i da mogu „umrijeti“. Međutim, ako se njima upravlja putem kontrolera više razine, oni se zamjenjuju kada se „pokvare“. Ali, zamjene dolaze s potpuno različitim IP adresama. To se također događa s operacijama uvođenja i skaliranja.

Kada zamjenjujemo stare Pod-ove novima, novi Pod-ovi dolaze s novim IP adresama. Na taj način dodaju se novi Pod-ovi s novim IP adresama, dok smanjenje uklanja postojeće Pod-ove.

Događaji poput ovih uzrokuju veliki odljev IP-a što se u nekim situacijama može pokazati kao veliki problem i taj detalj moramo imati na umu prilikom dizajniranja klastera i segmentacije interne mreže.



Slika 14. Mrežni model sustava Kubernetes

(Izvor: https://digitalvarys.com/kubernetes-networking-models/)

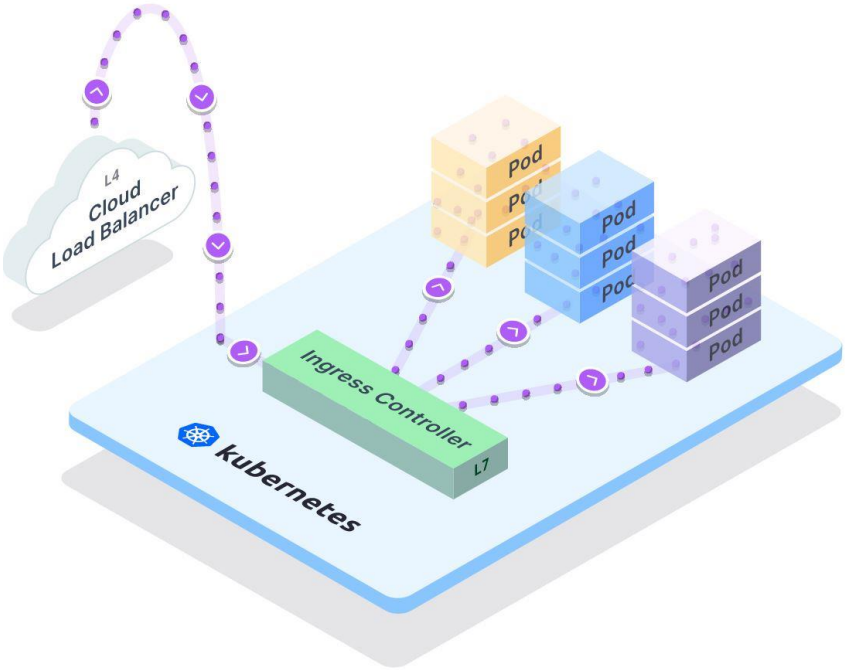
Međutim, kako Pod-ovi rade na TCP i UDP sloju, ne posjeduju inteligenciju aplikacija. To znači da ne mogu osigurati usmjeravanje hosta i na sloju aplikacije. Za to je potreban ***Ingress***, koji razumije HTTP i pruža usmjeravanje temeljeno na hostu i putu (eng. Path). Usluga ***Ingress*** donosi stabilne IP adrese i DNS imena u nestabilni svijet Podova.

Ingress kontroler

Ingress kontroler odnosi se na pristup više web aplikacija putem jednog *LoadBalance*r servisa. Pojam "***Ingress***" pisat će se velikim slovom jer je to također resurs u sustavu Kubernetes. *Ingress* izlaže HTTP i HTTPS rute izvan klastera uslugama unutar klastera. Usmjeravanje prometa kontroliraju pravila definirana na Ingress resursu. Može se konfigurirati tako da određenim uslugama daje vanjski dostupne URL-ove, balansira promet tj. opterećenje, terminira SSL/TLS te nudi virtualni hosting na temelju imena (eng. Name-based). *Ingress* kontroler odgovoran je za dolazni promet prema Kubernetes i to obično zajedno sa Load Balancer-om. Također, Ingress ne otkriva proizvoljne portove ili protokole.

Primjer YAML Ingress konfiguracije:

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: name-virtual-host-ingress
spec:
  rules:
    - host: web-app.com
      http:
        paths:
          - backend:
              serviceName: testsvc1
              servicePort: 80
    - host: app-web.com
      http:
        paths:
          - backend:
              serviceName: testsvc2
              servicePort: 80
```



Slika 15. K8s u oblaku sa Cloud LB-om ispred Ingress-a kontrolera
(Izvor: https://www.getambassador.io/docs/run/latest/kubernetes-network-architecture/)

Gornji primjer predstavlja tipičnu topologiju sustava Kubernetes kao usluge koju omogućuju/nude pružatelji usluga u oblaku (KaaS) kao što su Amazon Web Services, Azure, Digital Ocean ili Google GKE.

Pružatelji infrastrukture u oblaku olakšavaju stvaranje ovih LB-ova izravno iz Kubernetesa. To se postiže stvaranjem Kubernetes servisa tipa: „*LoadBalancer*“. Kada se ova usluga kreira, davatelj usluge koristit će metapodatke sadržane u definiciji Kubernetes servisa kako bi iste pružio prema LoadBalancer-u.

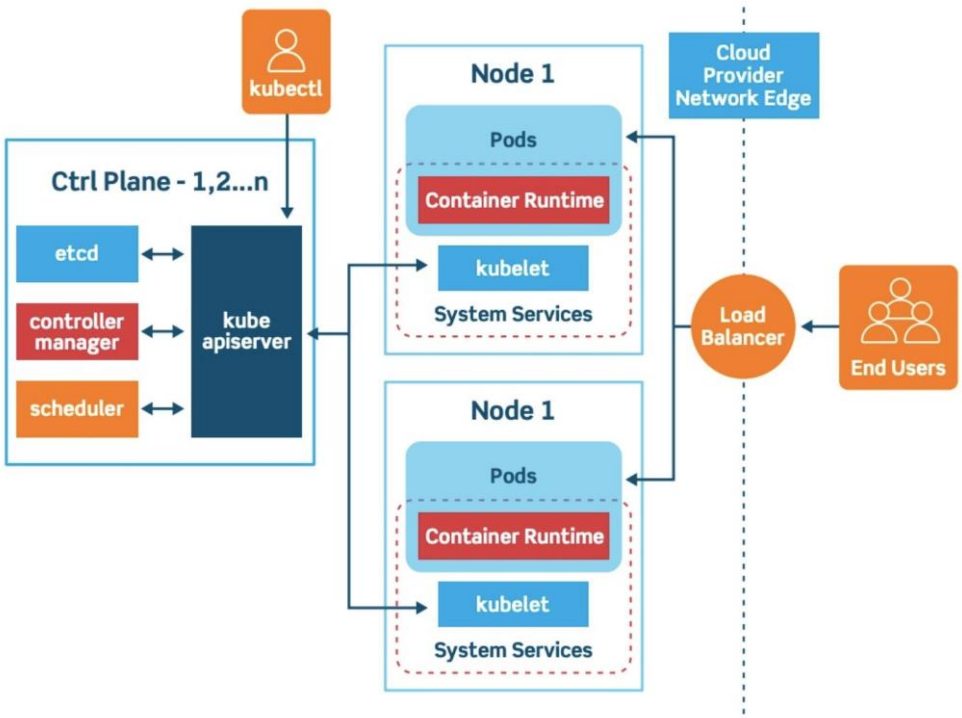
Iako postoji niz tehnika za usmjeravanje prometa u Kubernetes klaster, daleko najčešća i najpopularnija metoda uključuje implementaciju rubnih (eng. Edge) proxy servisa tj. Ingrees kontrolera unutar Kubernetes klastera zajedno s vanjskim *Load balancer* servisom koji je „de facto“ dio infrastrukture pružatelja usluge.

U većini situacija takova mrežna topologija puno je jednostavnija za implementaciju, ali pružatelji usluga dodatno naplaćuju uslugu *Load Balancera* što u nekim situacijama u kojima imamo potrebu izlaganja većeg broja aplikacija na Internet može u velikoj mjeri poskupiti samu uslugu. O tome trebamo posebno razmisliti prije samog zakupa KaaS usluge.

Nadalje, *LoadBalancer* je postavljen izvan Kubernetes klastera. Uobičajeno je da *LoadBalancer* ima jednu ili više statičkih IP adresa te prilikom konfiguracije koristimo ih kako bi smo unijeli DNS zapis za mapiranje naziva domene na statičku IP adresu (većina pružatelja usluga u oblaku ima i interni DNS servis na raspolaganju unutar kojeg možemo održavati DNS zapise).

Edge proxy ili *Ingress* je obično *Proxy* sloja 7 OSI modela koji se postavlja izravno u Kubernetes klaster. Osnovna funkcija rubnog proxy-ja tj. Ingress-a je prihvaćanje dolaznog prometa sa vanjskog balansera i usmjeravanje prometa prema Kubernetes uslugama.

Ingress bi se trebao/morao konfigurirati pomoću Kubernetes manifest datoteke u YAML formatu. To omogućuje zajedničko upravljanje konfiguracijama i za *Ingress* i za pripadajuću Kubernetes uslugu/e. *Ingress* je Kubernetes standard. Kao takav, to je resurs bez kojeg ne možemo zamisliti Kubernetes klaster.



Slika 16. Detaljnija arhitektura Kubernetes sustava sa Cloud Load Balancer-om
(Izvor: https://platform9.com/blog/kubernetes-enterprise-chapter-2-kubernetes-architecture-concepts/)

2.4 YAML programski jezik

„YAML je standard za serijalizaciju podataka prilagođen ljudima i za sve programske jezike."

YAML je čovjeku čitljiv jezik za serijalizaciju podataka koji se vrlo često koristi za pisanje konfiguracijskih datoteka.

Ciljevi YAML su čitljivost i što potpuniji informacijski model. YAML je složeniji za generiranje i raščlanjivanje, stoga se može promatrati kao prirodni nadskup JSON-a. Svaka JSON datoteka također je važeća YAML datoteka.

Kako se YAML koristi u sustavu Kubernetes? Resursi sustava Kubernetes kreirani su na deklarativni način, koristeći YAML datoteke. Kubernetes resursi, kao što su Pod-ovi, usluge, manifesti i implementacije kreiraju se pomoću YAML datoteka.

YAML je stvoren posebno za uobičajene slučajeve upotrebe kao što su:

- Konfiguracijske datoteke
- Log datoteke
- Među procesna razmjena poruka
- Dijeljenje podataka
- Postojanost objekta
- Složene strukture podataka

Zašto bi smo trebali koristiti YAML?

Postoji nekoliko prednosti korištenja YAML datoteka:

- Lako su čitljivi. YAML datoteke su jasne i proširive
- Jednostavne su za implementaciju i korištenje
- Lako su prenosive između programskih jezika
- Odgovaraju izvornim strukturama podataka
- YAML datoteke imaju dosljedan model za podršku generičkim alatima
- Podržavaju obradu u jednom prolazu
- Jednostavne su za održavanje
- Imaju mogućnost stvaranja složenije strukture nego što možemo koristiti kroz CLI

Kubernetes radi na temelju definiranog stanja i stvarnog stanja. Kubernetes objekti predstavljaju stanje klastera i govore sustavu Kubernetes kako želite da radno opterećenje izgleda. Resursi sustava Kubernetes, kao što su Pod-ovi, objekti i implementacije, mogu se kreirati pomoću YAML datoteka.

Prilikom izrade Kubernetes objekta, morat ćemo uključiti specifikacije za definiranje željenog stanja objekta. Kubernetes API može se koristiti za stvaranje objekta. Zahtjev za API uključivat će specifikacije objekta u JSON-u, ali najčešće ćete kubectl-u dati potrebne informacije kao YAML datoteku. *Kubectl* će pretvoriti datoteku u YAML kada uputi API zahtjev.

Nakon što je objekt stvoren i definiran, Kubernetes radi kako bi osigurao da objekt uvijek postoji. Sistem administratori sustava Kubernetes specificiraju definirano stanje koristeći YAML (ili JSON) datoteke koje šalju Kubernetes API-ju. Kubernetes koristi kontroler za analizu razlike između novog definiranog stanja i stvarnog stanja u klasteru.

Primjer YAML konfiguracijske datoteke za Nginx *Deployment*:

```
apiVersion: apps/v1
 kind: Deployment
 metadata:
   name: nginx-implementacija
 spec:
   selector:
     matchLabels:
       app: nginx
   replicas: 3 # govori sustavu Kubernetes da pokrene 3 Pod-a
   template:
     metadata:
       labels:
         app: nginx
     spec:
       containers:
       - name: nginx
         image: nginx:1.14.2 # govori sustavu Kubernetes koji Image koristiti
         ports:
         - containerPort: 80 # govori sustavu Kubernetes po kojem portu da radi
```

U YAML datoteci za određeni Kubernetes objekt koji želimo stvoriti, morat ćemo postaviti vrijednosti za sljedeća polja:

- ***apiVersion*** - koju verziju Kubernetes API-ja koristimo za stvaranje ovog objekta
- ***kind*** - kakav objekt želimo stvoriti
- ***metadata*** - podaci koji pomažu jedinstveno identificirati objekt, uključujući niz naziva, UID i izborni prostor imena (Namespace)
- ***spec*** - kakvo stanje želimo za određeni objekt

2.5 Instalacije sustava Kubernetes

Postoje najmanje tri tipična načina za instalaciju sustava Kubernetes:

1. Kubernetes igralište (eng. Kubernetes Playground)
2. Kubernetes u oblaku (KaaS)
3. „Uradi sam“ instalacija (eng. Do-It-Yourself; DIY)

Kubernetes igrališta (eng. Kubernetes Playground)

Kubernetes igrališta (eng. Kubernetes Playground) najbrži su i najlakši način da postavimo sustav Kubernetes, ali definitivno nisu za produkcijska okruženja. Popularni primjeri uključuju Play with Kubernetes, Katakoda, Docker Desktop, Minikube, k3d i još mnoge druge. Navedene implementacije koriste se isključivo za učenje, testiranje i stjecanje neophodnog iskustva u održavanju sustava Kubernetes.

Kubernetes u oblaku (eng. Kubernetes as a Service; KaaS)

Sve glavne platforme u oblaku nude hostiranu Kubernetes uslugu (KaaS). Ovo je model u kojem predajemo veliku odgovornost za Kubernetes infrastrukturu svom pružatelju usluga oblaka tj. on (pružatelj usluga u oblaku) obično se brine o stvarima poput visoke dostupnosti (HA), performansi i ažuriranja i općenito funkcioniranja sustava Kubernetes kao usluge.

Naravno, nisu sve hostirane Kubernetes usluge jednake. Iako davatelj usluge u oblaku upravlja velikim dijelom infrastrukture umjesto nas, konačna odgovornost ipak ostaje na nama.

Bez obzira na prednosti i nedostatke, hostirani sustav Kubernetes je veoma blizu „originalnom“ Kubernetes klasteru, ali sa ne toliko napora.

Kubernetes upravljački plan i radni čvorovi/nodeovi, s najboljim sigurnosnim praksama „out-of-the-box“, servisnom mrežom i još mnogo toga. I sve to u samo nekoliko jednostavnih klikova. Da, tako se to „reklamira“, ali većina navedenog ipak ne stoji jer nakon osnovne instalacije potrebno je odraditi još dosta posla kako bi sustav zadovoljio npr. samo osnovne sigurnosne standarde i to u većini slučajeva!

Popis najpopularnijih pružatelja usluga KaaS (eng. Kubernetes as a Service):

- AWS: Elastic Kubernetes Service (EKS)
- Azure: Azure Kubernetes usluga (AKS)
- Google Cloud Platform: Google Kubernetes Engine (GKE)
- Linode: Linode Kubernetes Engine (LKE)
- DigitalOcean: DigitalOcean Kubernetes (DOKS)
- Civo Cloud Kubernetes

Prije odluke o postavljanju Kubernetes na nekom od javnih oblaka potrebno je ozbiljno razmotriti i proanalizirati sve opcije i dimenzionirati sustav na ispravan i kvalitetan način s obzirom na stvarne potrebe, budget i opcije koje su na raspolaganju.

Jer, kao što je često slučaj s uslugama u oblaku, lako je prikupiti velike račune, ako npr. zaboravimo isključiti neku zakupljenu infrastrukturu ili uslugu koja se u određenom trenutku ne koristi ili kada ju više naprosto ne trebamo.

„Uradi sam“ Kubernetes klaster (eng. Do-It-Yourself ; DIY)

Daleko najteži način za postavljanje Kubernetes klastera je da ga sami postavimo na svojoj vlastitoj infrastrukturi tj. u vlastitom podatkovnom centru što je u većini slučajeva vrlo ozbiljan i zahtjevan projekt kojem bi se trebalo/moralo pristupiti sa velikim respektom.

Da, DIY instalacije u današnje vrijeme su puno lakše tj. jednostavnije nego što su bile, ali i dalje mogu biti teške, kompliciranе i zahtijevaju određena i to vrlo specifična znanja i vještine.

Međutim, DIY instalacije pružaju najveću fleksibilnost i kontrolu, što je svakako dobro. Uz to, ne iziskuju nikakve dodatne troškove što isto tako moramo uzeti u obzir i tretirati tu činjenicu kao prednost.

U slijedećem poglavlju, **3. Sigurnosti sustava za orkestraciju Kubernetes** opisati će se osnovne sigurnosne aspekte sustava Kubrnetes kao što su modeli prijetnji u sustavu Kubernetes, načelo najmanje privilegije, autentifikaciju, autorizaciju i *RBAC* model, sigurnost Kubernetes klaster komponenti, sigurnost kontejnera i Pod-ova, sigurnost *Secret* objekata te sigurnost mreže sustava Kubernetes.

3 SIGURNOSTI SUSTAVA ZA ORKESTRACIJU KUBERNETES

Imajući na umu model prijetnji, osnovne sigurnosne pretpostavke sustava Kubernetes možemo promatrati pomoću slijedećih glavnih i nezaobilaznih odrednica:

1. Autentifikacija i autorizacija
2. Kontrola pristupa temeljenu na ulogama (RBAC)
3. Sigurnost resursa sustava Kubernetes
4. Sigurnost mreže sustava Kubernetes
5. Sigurnost Pod-ova sustava Kubernetes
6. Nadzor, zapisivanje i revizija
7. Sigurnost kontejner Image-a i pripadajućih repozitorija
8. Periodična ažuriranja sustava Kubernetes

Takav pristup potreban je kako bi se osiguralo da su svi klaster node-ovi i sve glavne sastavnice sustava sigurne te da je površina napada svedena na minimum. Najbolji primjeri iz prakse koji proizlaze iz svakog od gornjih načela primjenjuju se na bilo koju implementaciju sustava Kubernetes, bilo da se koriti Kubernetes klaster u vlastitom podatkovnom centru ili kao uslugu u oblaku.

Treba svakako napomenuti da postoje povezane sigurnosne kontrole izvan sustava Kubernetes ili sigurnosni nadzor, koji mogu pomoći u smanjenju vjerojatnosti napada i povećanju obrambenog položaja.

Jako je važno promišljati o sigurnosti tijekom cijelog životnog ciklusa aplikacije umjesto da se usko fokusiramo na implementaciju kontejnera u sustav Kubernetes.

Međutim, radi kratkoće i vrlo opsežne teme, u ovom ćemo specijalističkom radu opisati tj. pokriti samo one sigurnosne kontrole koje se nalaze unutar neposrednog Kubernetes okruženja i to iz perspektive sistemskog održavanja sustava ne ulazeći u detalje koje se odnose na sigurnost poslova/procesa vezanih uz razvoj programskih rješenja, sigurnost kontejnera tj. aplikacija i CI/CD model.

Autentifikacija i autorizacija

Kubernetes API središnja su sučelja za administratore, korisnike i aplikacije za rad i komunikaciju u Kubernetes okruženju. I korisnici i računi mogu pristupiti Kubernetes API-jima za pokretanje raznih operacija. Kao takva, kontrola pristupa API-ju glavni je zadatak provjere autentičnosti i autorizacije unutar sustava Kubernetes.

Sustav Kubernetes ima ugrađene kontrole autentifikacije i autorizacije, kao i kontrole pristupa, koje presreću i reguliraju metode koje se mogu koristiti za pozivanje vanjske logike.

Autentifikacija i autorizacija su srž sigurnosti sustava Kubernetes i na taj detalj moramo uložiti dovoljno vremena kako bi smo sustav doveli do željene razine sigurnosti slijedeći primjere dobre prakse koji u većini slučajeve nisu specifični isključivo za Kubernetes sustav već čine skup aktivnosti i mjera karakterističnih za većinu IT sustava, aplikacija, servisa i sl.

Kontrola pristupa temeljena na ulogama (RBAC)

Kubernetes definitivno nije sigurnosna platforma niti je platforma koja podrazumijeva sigurnost prilikom ili neposredno nakon početne instalacije. U tom smislu sustavu Kubernetes nedostaju određeni „izvorni“ alat/i za rješavanje većine zadataka povezanih sa sigurnošću, kao što su npr. otkrivanje ranjivosti unutar aplikacija i praćenje kršenja sigurnosnih postavki.

Međutim, postoji jedan sigurnosni zadatak koji Kubernetes vrlo dobro rješava na nativni, tj. izvorni način, a to je RBAC (eng. Role Based Access Control) tj. kontrola pristupa temeljena na ulogama.

Kubernetes nudi opsežan i ugrađen RBAC okvir. Iskorištavanje prednosti Kubernetes RBAC osnovni je i prvi korak prema osiguranju klastera i aplikacija koje rade u sustavu Kubernetes.

Kontrola pristupa temeljena na ulogama ili RBAC osnovna je sigurnosna značajka sustava Kubernetes. To je također jedna od najboljih praksi implementacije sustava. Uz pomoć RBAC, možemo primijeniti pravila kontrole pristupa na postavljeni Kube API; ili pak možemo definirati kakvu vrstu dozvole treba dati korisnicima putem RBAC-a.

Iako je RBAC omogućen prema zadanim postavkama treba ga pravilno koristiti jer je to jedan od najosnovnijih primjera dobre prakse koji se odnose na sigurnost sustava Kubernetes.

Uglavnom, ne ulazeći previše u detalje, osnovna postavka RBAC je izbjegavati dupliciranje dopuštenja i ukloniti neiskorištene i neaktivne uloge. Na taj način sustav je postavljen i omogućuje nam da se usredotočimo samo na aktivne dijelove s time da pokušamo ne davati nepotrebna dopuštenja korisniku ukoliko ona nisu prijeko potrebna.

Izolacija resursa

Izolacija resursa je još jedna velika sigurnosna poluga unutar sustava Kubernetes. Izolacija ne samo da sprječava napade uskraćivanja usluge, već također osigurava privatnost i zaštitu podataka. Kubernetes platforma pruža izolacijske mehanizme za brojne tipove resursa, uključujući *Pod-ove i Namespaces*.

Ograničenja resursa koja možete postaviti na *Pod-ove* i *Namespaces* uključuju cikluse središnje procesorske jedinice (CPU), zahtjeve za memorijom i trajni prostor za pohranu.

Sigurnost mreže sustava Kubernetes

Učvršćivanje okoline, uključujući učvršćivanje kontejnera i temeljne Kubernetes infrastrukture ključno je za sigurnost Kubernetes klaster okruženja. Pomaže u obrani od prijetnji koje donose kompromitirani spremnici, zlouporabe i pogrešne konfiguracije.

Operacije učvršćivanja uključuju npr. ograničenja pokretanja privilegiranih kontejnera, ograničavanje eskalacije privilegija i može li kontejner pristupiti mrežnom sučelju glavnog (host) računala i njegovom datotečnom sustavu.

Mrežna sigurnost upravlja se segmentacijom mreže, osigurava pristup API-ju s provjerom autentičnosti klijenta, TLS (eng. Transport Layer Security) i upravlja popisima za kontrolu pristupa mreže (ACL-ovima). To je ustvari „mali“ vatrozid (eng. Firewall) na nivou npr. čvora.

Nadzor, zapisivanje i revizija (eng. Monitoring. Logging i Auditing)

Uz logiranje izvornih aplikacija i sustava, vrlo korisno je imati zapisane Kubernetes operacije, kao što je npr. zapis tko je pristupio kojem Kubernetes API-ju. Kubernetes nudi značajku *Audit i Logging* za obavljanje odvojenih funkcija bilježenja i revizije (eng. Audit).

Zapisivanje revizije bilježi radnje koje poduzima API. Zapisi se zatim mogu arhivirati za kasniju analizu. Administrator može odrediti koji se događaji trebaju zabilježiti kreiranjem YAML konfiguracijske datoteke politike revizije.

Sigurnost čvorova (eng. Master ang Worker Nodes)

Da bi se kontejneri pokrenuli na siguran način, svaki Linux čvor mora biti pravilno konfiguriran i ojačan. Centar za internetsku sigurnost (https://www.cisecurity.org/) i odgovarajuća CIS mjerila za Kubernetes sadrže mnoge smjernice za jačanje sigurnosti sustava Kubernetes koje bi operativni/sigurnosni timovi trebali slijediti.

Općenito, glavna razmatranja sigurnosti čvorova uključuju:

- Osigurati komunikaciju čvora s TLS klijentskim certifikatom, ili osigurati da su sve kritične API pristupne točke osigurane „sa kraja na kraj“ (eng. end-to-end) TLS-om,
- Omogućiti relevantne sigurnosne kontrole na razini Kernela kao što je SELinux. Ove mogućnosti pomažu u ograničavanju površine napada na čvor, dajući tako veću kontrolu nad sigurnošću cijelog sustava,
- Ograničiti izravni pristup, npr. pristupa SSH (eng. Secure Shell), Kubernetes čvorovima,
- Slijediti najbolje prakse, kao što je CIS Kubernetes Benchmark, sa ciljem ispravne konfiguracije Linux čvorova kao sastavnog dijela Kubernetes klastera.

Sigurnost kontejner Image-a

Jedan od aspekta sigurnosti sustava Kubernetes jest sigurnost slike kontejnera (eng. Container Image) i upravljanje ranjivostima. Budući da pokretanje kontejnera s ranjivostima izlaže sustav napadima moramo aktivno upravljati slikama koje se koriste u sustavu kako bismo otkrili i uklonili poznate ranjivosti. Posebice ukoliko se radi na način da se preuzimaju javno dostupni Images.

Brojni komercijalni i open source alati mogu izvršiti skeniranje slike kontejnera kako bi otkrili poznate CVE identifikatore uobičajenih ranjivosti i izloženosti. Trik je u tome da se ne zaustavimo samo i isključivo na skeniranju Image-a.

Umjesto toga, funkcija skeniranja trebala bi biti integrirana s mogućnostima provedbe i sanacije tijekom izvođenja.

Upotreba *Namespace* za organizaciju Kubernetes klastera

Jedna od najboljih praksi za poboljšanje sigurnosti Kubernetesa i otključavanje dodatnih mogućnosti je korištenje *NameSpace* za svojevrsno „particioniranje“ ili „izolacija“ dijela Kubernetes klastera.

NameSpace pomažu da Kubernetes klaster bude zaštićen od drugih timova koji rade na istom klasteru – ne osiguravajući da ovo razdvajanje čini mogućim slučajne smetnje i prepisivanje od strane inače dobronamjernih korisnika. Za svaki razvojni tim treba izraditi zasebne *NameSpace,* uključujući timove za razvoj, testiranje i produkciju, jer to smanjuje mogućnost nenamjernog djelovanja suprotno sigurnosnim postavkama.

Osim toga, *Namespace* omogućuju definiranje ograničenja resursa za Pod-ove koji se nalaze i izvode u istom. To pomaže u sprječavanju DoS (eng. Denial of Service) situacija uzrokovanih neprovjerenim i neispravnim skaliranjem resursa na nivou *Namaspace.*

Osigurati da klaster radi na najnovijoj stabilnoj verziji sustava Kubernetes

Koliko god se činilo očito, pokretanje najnovije verzije sustava Kubernetes najjednostavniji je način poboljšanja sigurnosti klastera koji se u velikom broju slučajeva naprosto zaboravlja.

Nova izdanja sadrže sigurnosne zakrpe, dodatne značajke i ažuriranja koja pomažu u smanjenju ranjivosti i doprinose većoj sigurnosti cjelokupnog sustava.

Pouzdan izvor informacija o Kubernetes sigurnosnim ranjivostima može se pronaći u CVE bazi podataka na adresi: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=kubernetes

3.1 Modeli prijetnji u sustavu Kubernetes

Sustav Kubernetes je kompleksan sustav koji se sastoji od više komponenti od kojih su najvažniji slijedeći:

1. ***Kube-apiserver,***
2. ***Etcd,***
3. ***Kube-scheduler,***
4. ***Kubelet.***

U postavljenoj (zadanoj) konfiguraciji između komponenti Kubernetes sustava kao rezultat mogu biti prijetnje kojih bi razvojni programeri i administratori klastera trebali/morali biti svjesni. Dodatno, implementacija aplikacija u sustav Kubernetes uvodi nove entitete s kojima aplikacija komunicira, dodajući nove aktere prijetnji i površine napada.

Cilj modeliranja prijetnji tj. detektiranja istih je pomoći u konfiguraciji, postavljanju, razvoju i administraciji kako bi shvatili da zadana Kubernetes konfiguracija nije dovoljna za zaštitu implementirane aplikacije od potencijalnih napadača.

Ovo poglavlje ima za cilj istaknuti prijetnje u Kubernetes sustavu, koji uključuje Kubernetes komponente i radna područja (eng. Workloads) u Kubernetes klasteru, kako bi administratori Kubernetes sustava, programeri i *DevOps* inženjeri razumjeli rizike svojih implementacija i imali plan za smanjenje rizika za poznate prijetnje.

Uvodno o modeliranju prijetnji

Modeliranje prijetnji je proces analize sustava kao cjeline tijekom faze dizajna ciklusa razvoja softvera (eng. Software Development Life Cycle; SDLC) kako bi se proaktivno identificirali rizici za sustav. Isto tako, modeliranje prijetnji koristi se i za analizu o sigurnosnim zahtjevima u razvojnom ciklusu kako bi se smanjila ozbiljnost rizika od samog početka razvoja aplikacije.

Modeliranje prijetnji uključuje identificiranje prijetnji, razumijevanje učinaka svake prijetnje i konačno, razvijanje strategije ublažavanja za svaku prijetnju.

Modeliranje prijetnji ima za cilj istaknuti rizike u ekosustavu kao jednostavnu matricu s vjerojatnošću i učinkom rizika i odgovarajuće strategije za smanjenje rizika ako postoji.

Nakon završenog modeliranja prijetnji, potrebno je definirati sljedeće:

1. **Sredstva:** Svojstvo ekosustava koje trebate zaštititi,
2. **Sigurnosna kontrola:** Svojstvo sustava koje štiti imovinu od identificiranih prijetnji i rizika. To su ili zaštitne mjere ili protumjere protiv rizika za imovinu,
3. **Akter prijetnje:** Akter prijetnje je entitet ili organizacija uključujući scenariste, napadače i haktiviste koji iskorištavaju određene ranjivosti i rizike,
4. **Površina napada:** Dio sustava s kojim akter prijetnje komunicira. Uključuje ulaznu točku aktera prijetnje u sustav,
5. **Prijetnja:** Rizik za određenu imovinu,
6. **Ublažavanje:** Ublažavanje definira kako smanjiti vjerojatnost i utjecaj prijetnje na imovinu.

IT industrija obično slijedi jedan od sljedećih pristupa modeliranju prijetnji:

- **STRIDE:** model STRIDE objavio je Microsoft 1999. godine. To je akronim za prijevaru, neovlašteno mijenjanje, odbijanje, otkrivanje informacija, uskraćivanje usluge i eskalaciju privilegija. STRIDE modelira prijetnje sustavu kako bi odgovorio na pitanje: „Što može poći po zlu sa sustavom?“
- **PASTA:** Proces za simulaciju napada i analizu prijetnji je pristup modeliranju prijetnji usmjeren na rizik. PASTA slijedi pristup usmjeren na napadače, koji koriste poslovni i tehnički timovi za razvoj strategija ublažavanja usmjerenih na imovinu.
- **VAST:** vizualno, agilno i jednostavno modeliranje prijetnji ima za cilj integrirati modeliranje prijetnji kroz razvoj aplikacija i infrastrukture s SDLC-om i agilnim razvojem softvera. Omogućuje vizualizacijsku shemu koja svim dionicima kao što su programeri, arhitekti, sigurnosni stručnjaci i poslovni rukovoditelji daje djelotvorne rezultate.

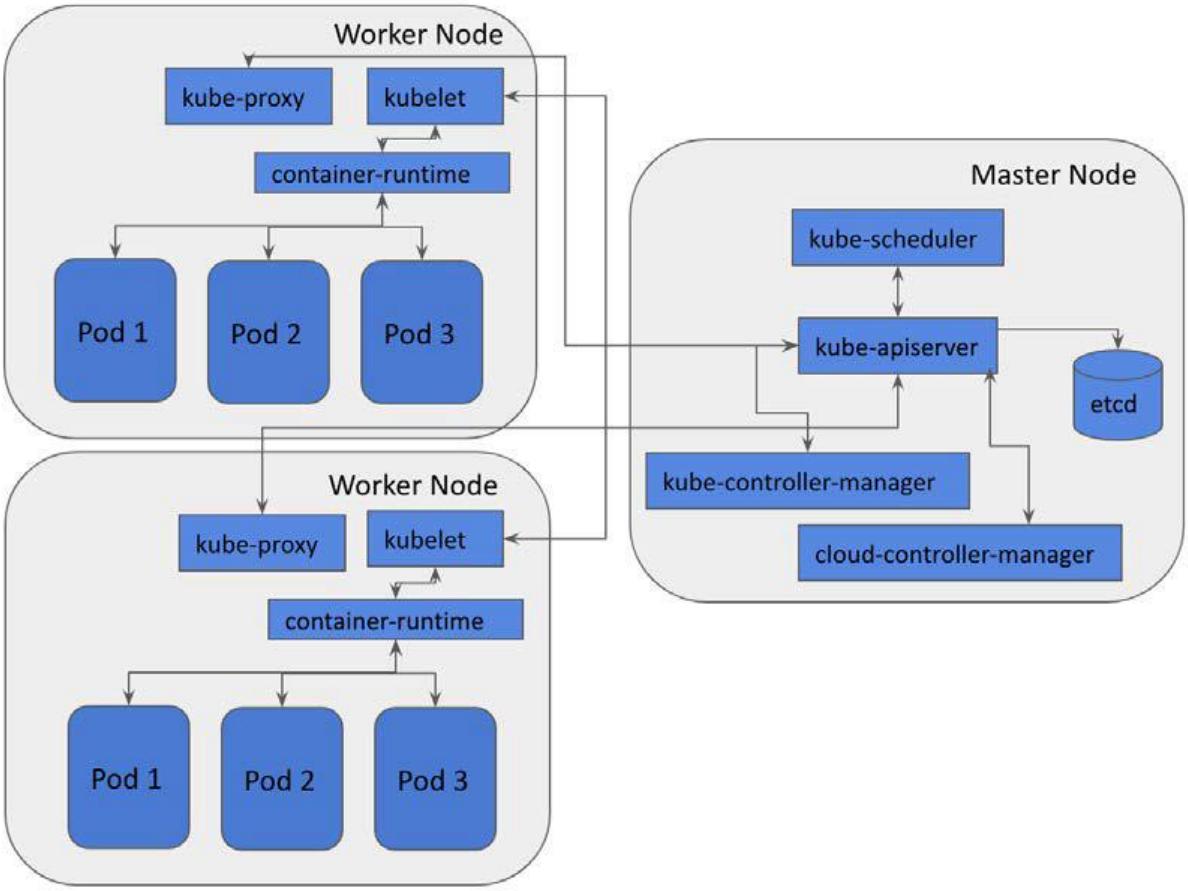
Postoje i drugi pristupi modeliranju prijetnji, ali prethodno navedena tri su najčešće korištena u IT industriji. Modeliranje prijetnji može biti beskonačno dug zadatak ako opseg modela prijetnji nije dobro definiran. Prije no što počne identifikacija prijetnji u ekosustavu, važno je da se jasno razumiju arhitektura i rad svake komponente, te interakcije između komponenti.

U uvodnim poglavljima već smo opisali funkcionalnost osnovnih Kubernetes komponenti. U nastavku prikazat će se interakcije između različitih komponenti u Kubernetes sustavu prije nego što se prikažu potencijalne prijetnje unutar Kubernetes sustava.

Interakcije komponenti sustava Kubernetes

Komponente sustava Kubernetes rade u suradnji kako bi osigurale da mikroservisi koji se izvode unutar klastera funkcioniraju prema očekivanjima. Ako se mikroservis implementira kao *DaemonSet,* Kubernetes komponente će se pobrinuti da postoji jedan modul koji pokreće mikroservis na svakom čvoru.

Slika 17. prikazuje interakciju komponenti na osnovnoj, visokoj razini funkcioniranja sustava Kubernetes.



Slika 17. Interakcija komponenti na osnovnoj razini funkcioniranja sustava Kubernetes.

Slijedi kratki opis što ove osnovne komponente sustava Kubernetes rade:

- ***Kube-apiserver:*** Kubernetes API poslužitelj (kube-apiserver) je komponenta kontrolne razine koja provjerava i konfigurira podatke za objekte,
- ***ETCD:*** etcd je baza podataka visoke dostupnosti koja se koristi za pohranu podataka kao što su konfiguracije, stanja i metapodaci i sl.,
- ***Kube-scheduler:*** *kube-scheduler* je zadani planer za sustav Kubernetes. Prati novostvorene Pod-ove i dodjeljuje Pod-ove čvorovima,
- ***Kube-controller-manager:*** Upravitelj Kubernetes kontrolera je kombinacija jezgrenih kontrolera koji prate ažuriranja stanja i u skladu s tim unose promjene u klaster,
- ***Cloud-controller-manager:*** Upravitelj cloud kontrolera pokreće kontrolere za interakciju s temeljnim pružateljima cloud usluga. (ova komponenta karakteristična je za implementacije sustava Kubernetes u okruženjima u oblaku i ne ulazi u opseg ovog specijalističkog rada),
- ***Kubelet:*** kubelet registrira čvor s API poslužiteljem na kontrolnoj razini i nadzire Pod-ove stvorene pomoću *podspecs* kako bi osigurao da su Pod-ovi i spremnici (eng. Containers) „zdravi“.

Ovdje je potrebno napomenuti da samo *kube-apiserver* komunicira s *etcd* bazom. Druge Kubernetes komponente kao što su *kube-scheduler, kube-controller-manager* i *cloud-controller manager* komuniciraju s *kube-apiserverom* koji radi na glavnim čvorovima (eng. MasterNode) kako bi ispunili svoje zadaće i odgovornosti. Na radnim čvorovima (eng. WorkerNode) i *kubelet* i *kube-proxy* komuniciraju s *kube-apiserverom.*

Da bi se kreirao *DaemonSet*, odrađuju se sljedeći koraci:

1. Korisnik šalje zahtjev prema *kube-apiserver*-u za stvaranje *DaemonSet* radnog okruženja putem HTTPS-a,
2. Nakon provjere autentičnosti, autorizacije i provjere valjanosti objekta, *kube-apiserver* stvara informacije o objektu radnog opterećenja za *DaemonSet* u bazi podataka *etcd.* Ni podaci u prijenosu ni u mirovanju nisu šifrirani prema zadanim postavkama sustava,
3. *DaemonSet* kontroler prati da se kreira novi *DaemonSet* objekt, a zatim šalje zahtjev za stvaranje Pod-a *kube-apiserveru*. Moramo znati da *DaemonSet* u osnovi znači da će se mikroservis izvoditi unutar modula na svakom čvoru,
4. *kube-apiserver* ponavlja radnje u koraku 2. i stvara informacije o objektu radnog opterećenja (eng. Workloads) za Pod-ove u bazi podataka *etcd*.
5. *kube-scheduler* promatra kako se kreira novi pod, zatim odlučuje na kojem čvoru pokrenuti pod na temelju kriterija odabira čvora. Nakon toga, *kube-scheduler* šalje zahtjev *kube-apiserveru* za čvor na kojem će se pod izvoditi.
6. *kube-apiserver* prima zahtjev od *kube-scheduler*-a i zatim ažurira *etcd* s informacijama o dodjeli čvora Pod-u.
7. *Kubelet* koji se izvodi na radnom čvoru promatra novi pod koji je dodijeljen ovom čvoru, zatim šalje zahtjev komponenti sučelja za vrijeme izvođenja spremnika (*CRI; Container runntime*) za pokretanje spremnika. Nakon toga, *kubelet* će poslati status modula natrag na *kube-apiserver*.
8. *kube-apiserver* prima informacije o statusu Pod-a od *kubelet* na ciljnom čvoru, zatim ažurira *etcd* bazu podataka sa statusom Pod-a.
9. Nakon što su moduli (iz *DaemonSet*-a) stvoreni, moduli mogu komunicirati s drugim Kubernetes komponentama i, ako je sve odrađeno kako treba, mikroservis bi trebao biti pokrenut.

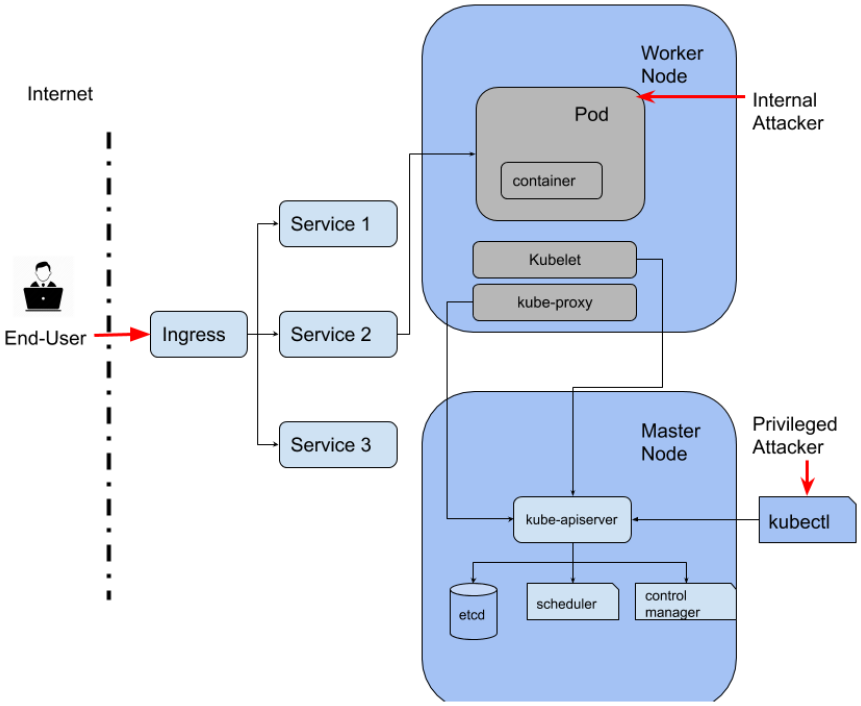
Prema zadanim postavkama sustava Kubernetes, komunikacija između gore navedenih i opisanih komponenti nije sigurna. Sigurnosne postavke ovise o konfiguraciji navedenih komponenti koje će se opisati u nastavku ovoga specijalističkog rada.

Analiza aktera prijetnji sustavu Kubernetes

Akter prijetnje je entitet ili maliciozni kod koji se izvršava u sustavu i od kojeg bi sustav (ili neka njegova komponenta, entitet i sl.) trebao biti zaštićen. Sa stajališta obrane, vrlo je važno razumjeti tko su potencijalni „neprijatelji“. U protivnom, obrambena strategija biti će previše nejasna, a samim time i neučinkovita. Akteri prijetnji u Kubernetes sustavima / okruženjima mogu se općenito klasificirati u tri kategorije:

1. **Krajnji korisnik:** Entitet koji se može povezati s aplikacijom. Ulazna točka za ovog aktera obično je balanser opterećenja (eng. Load Balancer) ili ulaz. Ponekad Pod-ovi, spremnici ili *NodePortovi* mogu biti izravno izloženi internetu, dodajući više ulaznih točaka za krajnjeg korisnika.
2. **Interni napadač:** Entitet koji ima ograničen pristup unutar Kubernetes klastera. Zlonamjerni spremnici ili moduli stvoreni unutar klastera primjeri su internih napadača.
3. **Privilegirani napadač:** Entitet koji ima administratorski pristup unutar Kubernetes klastera. Administratori infrastrukture, kompromitirane instance *kube-apiservera* i zlonamjerni čvorovi unutar Kubernetes klastera primjeri su privilegiranih napadača.

Slika 18. prikazuje različite aktere u sustavu Kubernetes koji mogu predstavljati prijetnju, a s obzirom i u odnosu na osnovne komponente sustava.



Slika 18. Različiti akteri u sustavu Kubernetes koji mogu predstavljati prijetnju

Kao što možemo vidjeti na ovom dijagramu, krajnji korisnik općenito stupa u interakciju s HTTP/HTTPS rutama koje otkriva ulazni kontroler, balanser opterećenja ili određeni moduli.

Krajnji korisnik je najmanje privilegiran. Interni napadač s druge strane ima ograničen pristup resursima unutar klastera. Privilegirani napadač ima najviše privilegija i ima mogućnost modificiranja klastera. Ove tri kategorije napadača pomažu odrediti ozbiljnost tj. razinu prijetnje.

Prijetnja koja uključuje krajnjeg korisnika ima veću ozbiljnost u usporedbi s prijetnjom koja uključuje povlaštenog napadača. Iako se ove uloge čine izoliranima u dijagramu, napadač se može promijeniti iz krajnjeg korisnika u internog napadača korištenjem napada s ciljem povećanja privilegija.

Prijetnje u Kubernetes klasterima

Nakon definiranja i opisa osnovnih komponenti sustava Kubernetes i aktera prijetnji, započinje modeliranje prijetnji Kubernetes klastera.

Čvorovi i Pod-ovi su osnovni tj. temeljni Kubernetes objekti. Moramo znati da su te komponente imovina koju svakako treba zaštititi od prijetnji. Ugrožavanje bilo koje od tih komponenti moglo bi dovesti do sljedećeg koraka u napadu na sustav, kao što je eskalacija privilegija, a što kasnije potencijalno može uzrokovati veliku štetu.

Također, moramo znati da su *kube-apiserver* i *etcd* mozak i srce Kubernetes klastera. Ako bi bilo koja od tih komponenti bila kompromitirana, to bi značilo i potencijalni kraj za Kubernetes klaster/sustav.

Tablice 1., 2., 3. i 4. navode potencijalne prijetnje u osnovnoj tj. početnoj (zadanoj) konfiguraciji sustava Kubernetes. Tablice također naglašavaju kako programeri i administratori Kubernetes klastera mogu zaštititi svoju imovinu od navedenih prijetnji upotrebom određenih sigurnosnih kontrola te uspostavom strategija ublažavanja prijetnji tj. rizika.

Tablica 1. Modeliranje prijetnji i sigurnosnih kontrola - Kubernetes klaster komponente

IMOVINA	PRIJETNJA	SIGURNOSNA KONTROLA	STRATEGIJA UBLAŽAVANJA
Kube- apiserver	Nema zadane politike revizije. To sprječava forenzičku analizu nakon napada.	Audit policy	Omogući politiku revizije. Razina metapodataka preporučuje se u cijelom ekosustavu.
	Nema zadane politike revizije. To sprječava forenzičku analizu nakon napada.	Authentication/ authorization	Osigurati da *--anonymous-auth* nije postavljeno na *false.*
	Provjera autentičnosti zbog upotrebe samopotpisanih certifikata.	Enable client CA using *--client-ca-file*	Pratiti ulazne veze s *kube-apiserver*-om kako bi provjerili ima li kakovih anomalija.

etcd	Podaci prema zadanim postavkama nisu šifrirani dok miruju.	Encrypt data using *--encryption- provider-config*	Proslijedite konfiguracijski parametar --encryption-provider-configtokube-apiserver prema zadanim postavkama.
	Provjera autentičnosti nije omogućena prema zadanim postavkama.	Authentication/ authorization	Osigurajte da *--anonymous-auth* nije postavljeno na *false*.
	Može mu pristupiti bilo koja komponenta u ekosustavu Kubernetes	mTLS	mTLS will *reject all* mTLS će odbiti sve veze na etcd osim na kube-apiserver. mTLS generira CA certifikat i ključ. Također generira odgovarajući poslužiteljski i klijentski certifikat i parove ključeva.
Kube- scheduler	Može mu se pristupiti bilo kojom komponentom u ekosustavu Kubernetes.	N/A	Označiti svaku vezu s *kube-planerom* osim *kube-api poslužitelja* kao zlonamjernu.
Controller manager	Nedostatak izolacije komponente može dovesti do napada eskalacije privilegija.	N/A	N/A
	Upravitelji kontrolera rukuju *Secret* kao što su varijable okruženja, argumenti naredbenog retka i Kubernetes *Secrets*, ali svaka komponenta ima minimalnu zaštitu za te *Secrets* datoteke	Tajna rotacija i korištenje Kubernetes Secrets	Secret u sustavu Kubernetes pružaju standardizirani način rukovanja sa Secrets datoketama. Secrets bi trebale biti konfigurirane tako da budu šifrirane pri pokretanju.
Kubelet	*kubelet* zapisuje bootstrap certifikate na disk (file sistem) nešifrirane.	N/A	Certifikate treba izbrisati nakon što se klaster pokrene. Također trebala bi biti omogućena potpuna enkripcija diska.
	kubelet krajnje točke mogu se koristiti za kompromitiranje čvorova i spremnika. Ove krajnje točke nisu autentificirane.	Authentication/ authorization	Osigurati CA paket s kubeletom kako bi osigurali autentifikaciju.

Tablica 2. Modeliranje prijetnji i sigurnosnih kontrola - Container Runtime

IMOVINA	PRIJETNJA	SIGURNOSNA KONTROLA	STRATEGIJA UBLAŽAVANJA
Container Runtime	Nepostojanje izlaznih (egress) filtera za vrijeme izvođenja spremnika može uzrokovati dohvaćanje zlonamjernih spremnika u ekosustavu	Audit policy	Omogući politiku revizije. Razina metapodataka preporučuje se na nivou cijelog sustava

Tablica 3. Modeliranje prijetnji i sigurnosnih kontrola - Razina klaster čvorova

IMOVINA	PRIJETNJA	SIGURNOSNA KONTROLA	STRATEGIJA UBLAŽAVANJA
Klaster čvorovi	Ugroženi čvorovi mogu ubaciti module kernela kako bi ugrozili module	/etc/modprobe.d/ kubernetes-blacklist.cont	Crna lista (eng. Black list) modula jezgre može osigurati zaštitu kontejnera od kompromitacije
	Ranjive binarne datoteke i usluge na čvorovima mogu dovesti do ugrožavanja pod-ova i klastera.	Minimized OS	Trebalo bi koristiti minimizirane operativne sustave kao što je Alpine koji imaju minimalne binarne datoteke i biblioteke za podršku *runtime* spremnika
	Pristup Kubernetes podacima uključujući *kubeconfig i* privatne ključeve	N/A	Host Intrusion Detection System (HIDS) i File Integrity Monitoring (FIM) mogu pomoći ako se pristupa kubeconfig-u i privatnim ključevima

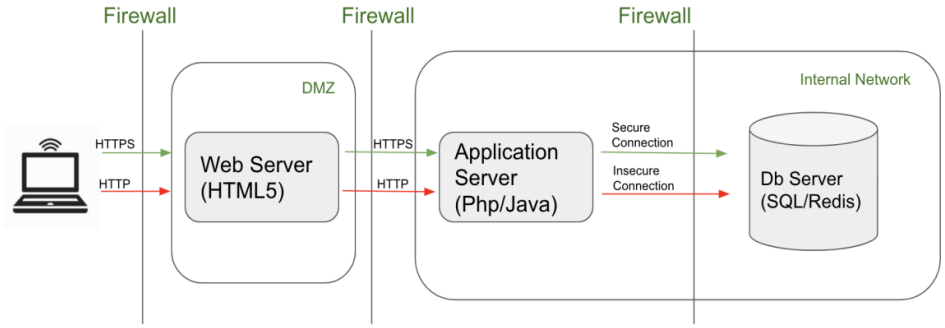
Tablica 4. Modeliranje prijetnji i sigurnosnih kontrola – Razina Pod-ova

IMOVINA	PRIJETNJA	SIGURNOSNA KONTROLA	STRATEGIJA UBLAŽAVANJA
Pod	Pod-ovi mogu raditi pod *root* korisnikom i kompromitirati host.	PodSecurity Policy	Pod-ovi rijetko trebaju *root* privilegije u redovnom tijeku rada. Politike sigurnosti Pod-ova mogu osigurati da se Pod-ovi ne pokreću kao povlašteni korisnici.
	Izolacija u Pod-ovima može se implementirati korištenjem mrežnih pravila.	N/A	Preporučljivo je testirati mrežni promet kako bi se provjerilo jesu li pravila ispravno primijenjena.
	Prema zadanim postavkama, svi su Pod-ovi povezani sa zadanim računom usluge. Ugrožene jedinice mogu pozvati APT pozive u klasteru, ako RBAC nije omogućen.	N/A	Kubectl zakrpa (eng. Patch) *serviceaccountdefault -p "automountServiceAccountToken: false*" Na taj način onemogućiti automatsko montiranje tokena za zadane servisne račune (eng. Service Account)

Modeliranje prijetnji u sustavu Kubernetes na primjeru jednostavne web aplikacije

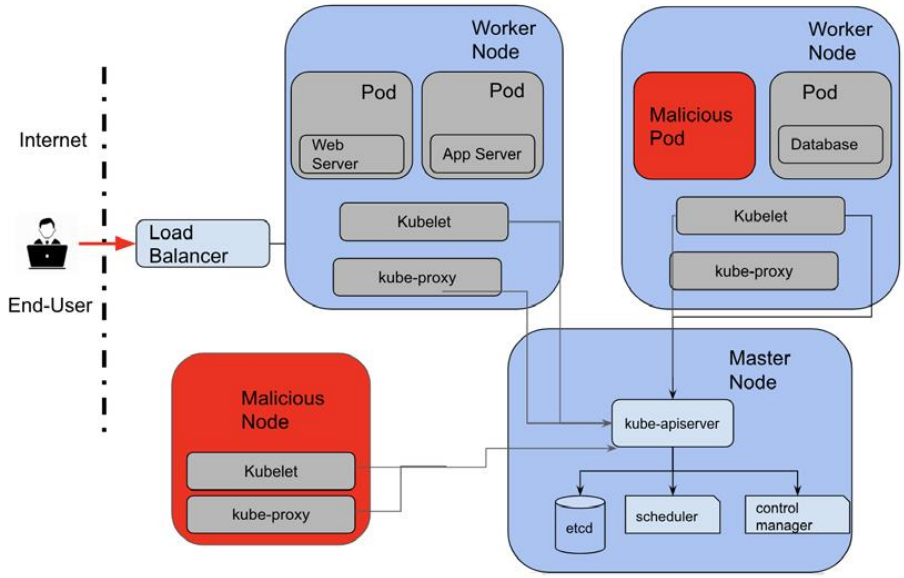
Modeliranje prijetnji uvelike se razlikuje za aplikaciju postavljenu u sustavu Kubernetes i aplikaciju postavljenu na „tradicionalan“ način. Implementacija u Kubernetes sustavu daje dodatnu složenost modelu prijetnji.

Kubernetes dodaje dodatna razmatranja, sredstva, aktere prijetnji i nove sigurnosne kontrole koje je potrebno razmotriti prije istraživanja prijetnji nad postavljenoj aplikaciji.



Slika 19. Model prijetnji nad web aplikacijom postavljenom na „tradicionalni“ način

Ista web aplikacija kao na **Slici 19.** izgleda ponešto drugačije u Kubernetes okruženju, a prikazana je na **Slici 20.**



Slika 20. Model prijetnji nad web aplikacijom postavljenom u sustavu Kubernetes

Kao što je prikazano na prethodnom dijagramu, web poslužitelj, aplikacijski poslužitelj i baza podataka rade unutar Pod-ova.

U **Tablicama 5., 6. i 7.** napravljena je usporedba modeliranja prijetnji između tradicionalne web arhitekture i arhitekture u sustavu Kubernetes na nivou imovine, aktera prijetnji i sigurnosnih kontrola.

Tablica 5. Modeliranje prijetnji na nivou imovine

	TRADICIONALNA WEB ARHITEKTURA	**ARHITEKTURA SUSTAVA KUBERNETES**
Imovina	Web server	Web server
	Aplikacijski server	Aplikacijski server
	Database server	Database server
	Čvorovi	Čvorovi (Master i radni)
		Persistent Volumes
		Kubernetes komponente (api-server,etcd,proxy, kubelet, scheduler, controller-manager)

Tablica 6. Modeliranje prijetnji na nivou aktera prijetnji

	TRADICIONALNA WEB ARHITEKTURA	**ARHITEKTURA SUSTAVA KUBERNETES**
Akteri prijetnji	Vanjski korisnici/napadači	Vanjski korisnici/napadači
	Interni korisnici/napadači	Interni korisnici/napadači
	Administratori sustava	Administratori sustava
		Maliciozni/kompromitirani čvorovi
		Maliciozni/kompromitirani Pod-ovi
		Kompromitirane Kubernetes komponente
		Aplikacije unutar klastera

Tablica 7. Modeliranje prijetnji na nivou sigurnosnih kontrola

	TRADICIONALNA WEB ARHITEKTURA	ARHITEKTURA SUSTAVA KUBERNETES
Sigurnosne kontrole	Vatrozid	Network policies
	DMZ	TLS, mTLS
	Interna mreža	Pod Security Policy
	Web aplikacijski vatrozid	Web aplikacijski vatrozid
	TLS konekcije	Pod izolacija
	File Encryption	File enkripcija
	Database autorizacija	Database autorizacija
	Database enkripcija	Database enkripcija
		Admission Controllers
		Kubernetes autorizacija

Kao zaključak, a vezano uz prethodnu usporedbu možemo zaključiti da Kubernetes sustav ima više imovine tj. komponenti koju je potrebno zaštititi.

S druge strane, sustav Kubernetes pruža više sigurnosnih kontrola, ali predstavlja i veću složenost tj. kompleksnost implementacije sigurnosti. No, više sigurnosnih kontrola ne znači nužno i veću sigurnost jer u većini slučajeva složenost i kompleksnost sustava je u pravilu „neprijatelj“ sigurnosti.

Drugim riječima, složenost i kompleksnost uvedena dizajnom sustava Kubernetes čini modeliranje prijetnji kompliciranijim u usporedbi sa tradicionalnom arhitekturom IT sustava.

Ono što je vrlo važno za napomenuti jest da modeliranje prijetnji može biti dug i složen proces, ali svakako ga vrijedi odraditi i pripremiti kako bi se što detaljnije razumjelo sigurnosno stanje cjelokupnog Kubernetes sustava.

3.2 **Načelo najmanje privilegije** (eng. Least Privilege)

Načelo najmanje privilegije kaže da bi svaka komponenta sustava trebala imati minimalan pristup podacima i resursima kako bi mogla funkcionirati. U okruženjima kao što je sustav Kubernetes, višestrukim resursima mogu pristupiti različiti korisnici i/ili objekti. Načelo najmanje privilegije osigurava da je šteta na klasteru minimalna, ako se korisnici ili objekti „loše“ ponašaju u takvim okruženjima.

S obzirom na složenost sustava Kubernetes, prvo će se analizirati Kubernetes objekti, a zatim privilegije dostupne za objekte. Zatim će se govoriti o privilegijama Kubernetes objekata i mogućim načinima njihovog ograničavanja.

U tom smislu, opisati će se različiti objekti sustava Kubernetes, kao što su *Namaspaces*, servisni računi, uloge i povezivanja uloga sa Kubernetes sigurnosnim značajkama, kao što su sigurnosni kontekst, *PodSecurityPolicy* i *NetworkPolicy*, koji se mogu iskoristiti za implementaciju načela najmanje privilegije za Kubernetes klaster.

Privilegija je ovlaštenje za izvođenje određene radnje kao što je pristup resursu ili obrada nekih podataka. Načelo najmanjih privilegija je ideja da svaki subjekt, korisnik, program, proces i sl. treba imati samo minimalne potrebne privilegije za obavljanje svoje funkcije. Ni više ni manje!

Na primjer, korisnik „tgolubic“, je običan korisnik određenog Linux stroja, može stvoriti datoteku u vlastitom matičnom direktoriju. Drugim riječima, korisnik „tgolubic“, ima privilegiju ili dopuštenje za stvaranje datoteke u svom početnom direktoriju. Međutim, korisnik „tgolubic“, neće moći stvoriti datoteku u imeniku drugog korisnika jer nema privilegiju ili dopuštenje za to.

Ako niti jedan od dnevnih zadataka korisnika „tgolubic“ zapravo ne koristi privilegiju za stvaranje datoteke u početnom direktoriju, a on ima privilegiju za to, tada administrator sustava mora revidirati prava za tog korisnika jer nije u skladu s načelom *najmanje privilegije*.

Koristi primjene načela najmanje privilegije

Iako bi moglo potrajati dosta vremena da se shvati koje su minimalne privilegije za subjekte kako bi obavljali svoje funkcije, dobiti od te strategije su značajne, pod pretpostavkom da je načelo najmanje privilegije implementirano u okruženju sustava Kubernetes, a to su:

- **Veća sigurnost:** Unutarnje prijetnje, širenje zlonamjernog softvera, bočno kretanje i sl. mogu se ublažiti implementacijom načela najmanje privilegije. Curenje informacija i podataka najčešće se događa zbog nedostatka načela najmanjih privilegija,
- **Bolja stabilnost:** s obzirom da su subjektima pravilno dodijeljene samo potrebne privilegije, aktivnosti subjekata postaju predvidljivije. Zauzvrat, stabilnost sustava je ojačana,
- **Poboljšana spremnost za reviziju:** s obzirom na to da su subjektima pravilno dodijeljene samo potrebne privilegije, opseg revizije će se dramatično smanjiti. Osim toga, mnogi uobičajeni sigurnosni standardi propisuju provedbu načela najmanjih privilegija kao uvjet usklađenosti.

Primjena najmanjih privilegija Kubernetes subjekata

Računi usluge Kubernetes, korisnici i grupe komuniciraju s *kube-apiserverom* za upravljanje Kubernetes objektima. Uz omogućen RBAC, različiti korisnici ili računi usluga mogu imati različite privilegije za upravljanje Kubernetes objektima. Na primjer, korisnici u grupi *system:master* imaju dodijeljenu ulogu administratora klastera, što znači da mogu upravljati cijelim Kubernetes klasterom, dok korisnici u grupi *system:kube-proxy* mogu pristupiti samo resursima potrebnim za *kube-proxy* komponentu.

3.3 Autentifikacija, autorizacija i RBAC model

Autentifikacija i autorizacija imaju vrlo važnu ulogu u osiguravanju sigurnosti aplikacija. Ova se dva pojma često koriste kao sinonimi, ali su vrlo različiti. Autentifikacijom se potvrđuje identitet korisnika. Nakon što je identitet potvrđen, autorizacija se koristi za provjeru ima li korisnik potrebne privilegije za obavljanje željene radnje.

Autentifikacija koristi nešto što korisnik zna za provjeru svog identiteta, a u najjednostavnijem obliku, to je korisničko ime i lozinka. Nakon što aplikacija potvrdi identitet korisnika, provjerava kojim resursima korisnik ima pristup. U većini slučajeva ovo je varijanta popisa kontrole pristupa. Popisi kontrole pristupa za korisnika uspoređuju se s atributima zahtjeva za dopuštanje ili odbijanje radnje.

Kubernetes autentifikacija

Svi zahtjevi u sustavu Kubernetes potječu od vanjskih korisnika, servisnih računa ili Kubernetes komponenti. Ako je podrijetlo zahtjeva nepoznato, tretira se kao anonimni zahtjev. Ovisno o konfiguraciji komponenti, anonimni zahtjevi mogu biti **dopušteni** ili **odbačeni** od strane modula za provjeru autentičnosti.

U verzijama sustava Kubernetes 1.6 pa na dalje, anonimni pristup dopušten je za podršku anonimnim i ne autentificiranim korisnicima za RBAC i ABAC načine autorizacije. Kubernetes koristi jednu ili više ovih strategija provjere autentičnosti.

Provjere autentičnosti mogu se implementirati na razne načine kao što su upotreba klijentskih certifikata, statičnih token-a, osnovne (eng. Basic) autentifikacije, upotrebom *„Bootstrap“* tokeen-a, token-a za servisne račune i sl.

Klijentski certifikati

Korištenje X509 Certificate Authority (CA) certifikata najčešća je strategija provjere autentičnosti u sustavu Kubernetes. Može se omogućiti prosljeđivanjem *--client-ca-file* prema poslužitelju.

Datoteka proslijeđena API poslužitelju ima popis CA-ova, koji stvara i potvrđuje klijentske certifikate u klasteru. Svojstvo zajedničkog imena u certifikatu često se koristi kao korisničko ime za zahtjev, a svojstvo organizacije koristi se za identifikaciju korisničkih grupa.

Kubernetes autorizacija

Autorizacija određuje je li zahtjev dopušten ili odbijen. Nakon što se identificira podrijetlo zahtjeva, aktivni autorizacijski moduli procjenjuju atribute zahtjeva u odnosu na pravila autorizacije korisnika kako bi dopustili ili odbili zahtjev. Svaki zahtjev uzastopno prolazi kroz autorizacijski modul i ako bilo koji modul donese odluku o dopuštanju ili odbijanju, automatski se zahtjev prihvaća ili odbija.

Atributi zahtjeva

Moduli za autorizaciju analiziraju skup atributa u zahtjevu kako bi odredili treba li zahtjev analizirati, dopustiti ili odbiti:

- ***User:*** Pokretač zahtjeva. Ovo se potvrđuje tijekom provjere autentičnosti,
- ***Group*****:** Grupa kojoj korisnik pripada. To je osigurano u sloju provjere autentičnosti,
- ***API:*** odredište zahtjeva,
- ***Verb:*** vrsta zahtjeva, koji može biti *GET, CREATE, PATCH, DELETE*, i sl.
- ***Resours****:* ID ili naziv resursa kojem se pristupa,
- ***Namespace:*** Imenski prostor resursa kojem se pristupa,
- ***Path:*** Ako je zahtjev za krajnju točku koja nije resurs, put se koristi za provjeru je li korisniku dopušten pristup krajnjoj točki.

Kubernetes omogućava nekoliko autorizacijskih modela kao što su: autorizacija čvora, ABAC, RBAC i sl. U nastavku ovog specijalističkog rada obraditi će se RBAC kao autorizacijski model temeljen na primjeni kontrole pristupa temeljene na ulogama (eng. Roles).

Autorizacijski model primjenom kontrole pristupa temeljene na ulogama (RBAC)

Kada govorimo u kontekstu primjene načela najmanjih privilegija, većinu vremena govorimo u kontekstu autorizacije. U različitim okruženjima primjenjuju se različiti modeli autorizacije.

Na primjer, popis kontrole pristupa (eng. Access Control List; ACL) uobičajeno se koristi u Linux OS-u i mrežnim vatrozidima (eng. Firewall), dok se RBAC koristi u sustavima baza podataka. Također je na administratoru sustava/okruženja da definira politike autorizacije kako bi se osigurale najmanje privilegije na temelju modela autorizacije dostupnih u sustavu.

RBAC tj. kontrola pristupa temeljena na ulogama temelji se na odluci o autorizaciji s obzirom na korisnikove uloge koje sadrže grupu dopuštenja ili privilegija. Na primjer, u Linux-u, korisnik se dodaje u različite grupe kako bi dobio pristup nekim mapama umjesto da mu se pojedinačno dodijeli pristup tim istim mapama u datotečnom sustavu (eng. File System).

Kao što je ranije spomenuto, RBAC je model reguliranja pristupa resursima na temelju uloga dodijeljenih korisnicima ili grupama. Od verzije sustava Kubernetes 1.6 pa na nadalje, RBAC je omogućen prema zadanim postavkama u sustavu Kubernetes. Prije verzije 1.6, RBAC se mogao omogućiti pokretanjem API poslužitelja (eng. Application Programming Interface) s oznakom *--authorization-mode=RBAC.*

RBAC olakšava dinamičku konfiguraciju pravila dopuštenja pomoću API poslužitelja. Temeljni elementi RBAC-a uključuju sljedeće:

1. ***Subject:*** Računi usluga, korisnici ili grupe koji traže pristup Kubernetes API-ju,
2. ***Resources:*** Kubernetes objekti kojima subjekt mora pristupiti,
3. ***Verbs:*** Različite vrste pristupa subjektu potrebnim resursu, npr. pregled, stvaranje, ažuriranje, popis, brisanje.

Zašto je RBAC važan?

RBAC mehanizmi nude granuliranu kontrolu korisničkih dozvola. Mogu se precizno dodijeliti značajke pojedincima kojima su potrebne, izbjegavajući pretjerano privilegirane račune o čemu se govorilo u prethodnom poglavlju.

Davanje previše mogućnosti korisnicima predstavlja rizik ako se vjerodajnice izgube ili ukradu. Na primjer, programeri vjerojatno ne bi trebali moći izbrisati produkcijske implementacije. Čak i ako bezuvjetno vjerujete svojim zaposlenicima, napad krađe identiteta ili društveni inženjering mogao bi ljudima izvan tvrtke dati kontrolu nad određenim „probijenim“ računom. Pomoću RBAC-a u sustavu Kubernetes mogu se stvoriti pravila koja sprječavaju korisnike da npr. izbrišu Pod-ove, u određenom *Namespace* i sl.

RBAC radi samo ako je aplikacija dizajnirana da ga podržava. Svaka logička sposobnost trebala bi imati svoju ulogu. Uloga se zatim može dodijeliti korisnicima uloge koje imaju smisla s obzirom na njihove odgovornosti te kako bi dobili željeni skup privilegija.

Korištenje RBAC-a u sustavu Kubernetes

Sustav Kubernetes ima opsežnu podršku za RBAC. Prožima arhitekturu sustava i podržava razgraničenje uloga prema *Resources* i *Verbs*. Na primjer, svaka od sljedećih radnji mogu se izraziti kao posebna RBAC pravila:

- Ispis Pod-ova,
- Kreiranje Pod-ova,
- Brisanje Pod-ova,
- Pregledavanje podataka unutar „Secrets“ datoteka,
- Brisanje implementacija,
- Dodavanje korisnika.

Pravila se kombiniraju u uloge koje pokrivaju više radnji. Gore prikazana pravila mogu se upotrijebiti za stvaranje uloge razvojnog programera koja može ispisivati i stvarati Pod-ove i zasebne administratorske uloge koja može ispisivati i stvarati Pod-ove, ali također mogu pokrenuti brisanje i upravljati korisnicima. Potrebno je imati preciznu kontrolu nad mogućnostima dostupnim svakom korisniku. To je od krucijalne važnosti!

Slijedeća četiri temeljna objekta čine Kubernetes RBAC implementacije:

- ***Role*****:** zbirka jednog ili više pravila koja definiraju akcije koje korisnici mogu poduzeti,
- ***RoleBinding:*** povezanost između uloge i jednog ili više korisnika ili servisnih računa, koji se nazivaju *subjekti*. Svaki predmet na koji se odnosi vezanje može obavljati radnje obuhvaćene ulogom,
- ***ClusterRole*** **i** ***ClusterRoleBinding:*** Ovo su varijante *Role* i *RoleBinding* koje postoje na razini klastera. Koriste se za stvaranje pravila koja ciljaju resurse na razini Kubernetes klastera kao što su npr. čvorovi i sl.

Odnos između korisničkih računa i servisnih računa

U sustavu Kubernetes RBAC politike mogu se koristiti za definiranje prava pristupa ljudskih korisnika (ili grupa ljudskih korisnika). Kubernetes identificira ljudske korisnike kao korisničke račune.

Međutim, RBAC pravila također mogu upravljati ponašanjem softverskih resursa, koje Kubernetes identificira kao servisne račune.

Iako sustav Kubernetes pravi razliku između korisničkih računa i servisnih računa na konceptualnoj razini, razlika zapravo nije bitna što se tiče RBAC pravila. To čini Kubernetes RBAC drugačijim od mnogih drugih RBAC sustava, koji se obično fokusiraju na upravljanje dozvolama ljudskih korisnika (na temelju čimbenika kao što su njihova radna uloga ili vrsta računa), a ne na upravljanje pristupom softverskim uslugama.

Odnos između *Roles* i *Cluster Roles*

Relativno jednostavna, ali važna karakteristika Kubernetes RBAC-a je ta što povlači razliku između dopuštenja koja se primjenjuju na resurse unutar jednog *Namespace*, kojima se upravlja putem Uloga, (eng. *Roles*) i onih koja se primjenjuju na cijeli klaster i kojima se upravlja putem Klaster uloga, (eng. *ClusterRoles).*

U većini slučajeva radit će se s ulogama koje se mogu koristiti za upravljanje dopuštenjima na detaljnijoj osnovi (tj. unutar pojedinačnih *Namespace*). Ali ponekad će se htjeti koristiti *ClusterRoles* za upravljanje pravilima za resurse koji postoje i na razini cijelog klastera.

Fleksibilne definicije resursa

Kubernetes RBAC također je prilično širok kada je u pitanju način na koji Kubernetes definira entitete kojima RBAC politike mogu upravljati.

Kubernetes te entitete naziva *Resources,* a oni mogu biti gotovo što god: Pod-ovi, dnevnici, ulazni kontroleri ili bilo koja druga vrsta prilagođenih resursa koje odlučimo definirati.

Većina drugih RBAC sustava ima tendenciju da bude restriktivnija u pogledu vrsta resursa kojima možete upravljati. Na primjer, IAM okviri u oblaku dizajnirani su za upravljanje samo unaprijed definiranim vrstama resursa, poput instanci virtualnog stroja ili spremnika za pohranu.

Upravljanje dozvolama putem "*Verbs*"

Za razliku od sustava kontrole pristupa koji mogu samo dopustiti ili odbiti pristup, ili sustava koji dijele prava pristupa u široke kategorije kao što su "čitanje", "pisanje" i "izvršavanje", Kubernetes RBAC pruža niz dozvola koji definiraju specifične radnje koje korisnički računi mogu obavljati nad određenim resursima.

Na primjer, možete dopustiti korisniku da "stvori" i "popiše" dani resurs navođenjem odgovarajućih postavki za „*Verbs*“ unutar pravila RBAC.

Na kraju ovog poglavlja navesti ćemo način na koji se kreiraju *kube-config* datoteke za pristup Kubernetes klasteri te kreiranje pripadajućih *Role* i *RoleBinding.* U primjeru ćemo prvo kreirati *privatni ključ* i *certifikat* za korisnika „mmatic“ koristeći „*openssl*“.

Nakon toga kreirati će se *kube-config* datoteka za korisnika „*mmatic*“ u *Namespace* pod nazivom „*test*“ u kojem će isti korisnik imati sva prava kao što su pregled, stvaranje, ažuriranje, popisivanje, brisanje … svih objekata u navedenom *Namespace.*

Na kraju kreirati će se *Role* i *RoleBinding* za navedenog korisnika.

Kreiranje *kube-config* datoteke i generiranje korisničkog certifikata i privatnog ključa izvršava se na Master čvoru na način:

```
# Generiranje privatnog ključa za korisnika „mmatic“ (mmatic.key)

[tgolubic@kube-master ~]$ openssl genrsa -out mmatic.key 2048

# Generiranje CSR-a za korisnika „mmatic“ (mmatic.csr user=mmatic; group=dev)

[tgolubic@kube-master ~]$ openssl req -new -key mmatic.key -out mmatic.csr -subj
"/CN=mmatic/O=dev"

# Kopiranje Kubernetes CA certifikata i ključa sa Master čvora

[tgolubic@kube-master ~]$ scp root@kmaster:/etc/kubernetes/pki/ca.{crt,key} .

# Potpisivanje certifikata sa Kubernetes CA i generiranje certifikata za „mmatic“

[tgolubic@kube-master ~]$ openssl x509 -req -in mmatic.csr -CA ca.crt -CAkey
ca.key -CAcreateserial -out mmatic.crt -days 365

# Kreiranje i spremanje kube-config datoteke za „mmatic“ na datotečni sustav hosta

[tgolubic@kube-master ~]$ kubectl --kubeconfig mmatic.config config set-cluster
kubernetes --server https://10.0.87.100:6443 --certificate-authority=ca.crt

[tgolubic@kube-master ~]$ kubectl --kubeconfig mmatic.config config set-
credentials mmatic --client-certificate /home/tgolubic/user-cert/mmatic.crt --
client-key /home/tgolubic/user-cert/mmatic.key
```

Pripremljena *kube-config datoteka* za korisnika „mmatic“ izgleda ovako:

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: # CA certifikat Kubernetes klastera
    server: https://10.0.87.100:6443 # Adresa i port Kubernetes Master čvora
  name: kubernetes # Ime/naziv Kubernetes klastera
contexts:
- context:
    cluster: kubernetes
    namespace: test # Namespace u kojem korisnik „mmatic“ ima prava pristupa
    user: mmatic
  name: mmatic-kubernetes
current-context: mmatic-kubernetes
kind: Config
preferences: {}
users:
- name: mmatic
  user:
    client-certificate-data: # Certifikat kojeg smo generirali za user „mmatic“
    client-key-data: # Privatni ključ kojeg smo generirali za user „mmatic“
```

Kako bi smo „zaokružili priču“ oko kreiranja *kube-config* datoteke za pristup Kubernetes klasteru na nivou *Namespace* pod nazivom „test“ za korisnika „mmatic“ i pripadajućih RBAC pravila potrebno je kreirati *Role* i *RoleBinding* datoteke i postaviti ih na Kubernetes klaster.

***Role* datoteka (u yaml formatu) za korisnika „mmatic“:**

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: mmatic-test-rola # Ime / naziv Role
  namespace: test # Namespace u kojem korisnik „mmatic“ ima prava pristupa
  resourceVersion: "611585"
  uid: 68878e00-7790-48a4-b85f-c29f1c81ff18
rules:
- apiGroups:
  - '*' # Sve API grupe
  resources:
  - '*' # Svi resursi
  verbs:
  - '*' # Sva prava
```

RoleBinding **datoteka (u yaml formatu) za korisnika „mmatic“:**

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: "2023-01-10T10:52:21Z"
  name: mmatic-test-rolebinding
  namespace: test # Na koji Namespace se odnosi
  resourceVersion: "611779"
  uid: b361c595-05f3-47c6-8d46-e1a05bcab917
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: mmatic-test-rola # Ime / naziv Role
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: mmatic # na kojeg user-a se odnosi
```

3.4 Sigurnost Kubernetes klaster komponenti

Kubernetes klaster sastoji se od nekoliko komponenti koje uključuju:

1. ***Kube-apiserver,***
2. ***ETCD bazu podataka,***
3. ***Kube-scheduler,***
4. ***CoreDNS,***
5. ***Kube-controller-manager***

Komponente glavnog čvora (eng. MasterNode) uključuju *Kube-apiserver, ETCD bazu podataka, Kube-scheduler, CoreDNS i Kube-controller-managerkubelet,* a komponente koje se nalaze na svim radnim čvorovima (eng. WorkerNodes) jesu *kubelet*, *kube-proxy i container-runtime*. Master komponente, tj. komponente koje se nalaze na glavnom čvoru odgovorne su za upravljanje samim Kubernetes klasterom.

Komponente glavnog čvora čine kontrolni prostor (eng. Control Plane) Kubernetes klastera. S druge strane, komponente radnih čvorova (pretpostavljamo da se Kubernetes klaster sastoji od više čvorova; jednog ili više glavnih čvorova i jednog ili više radnih čvorova) odgovorne su za funkcioniranje Pod-ova i pripadajućih spremnika na pojedinom čvoru.

U djelu specijalističkog rada u kojemu je obrađeno modeliranje prijetnji, navedeno je da komponente u Kubernetes klasteru moraju biti konfigurirane kako bi se osigurala sigurnost klastera. Iz toga proizlazi da kompromitiranje bilo koje komponente Kubernetes klastera može uzrokovati povredu i/ili ugrozu podataka, smetnje u radu klastera, probleme u radu aplikacija unutar pod-ova i pripadajućih spremnika, a u konačnici i pad cjelokupnog klastera i gubitak podataka.

Pogrešna konfiguracija klaster okruženja jedan je od primarnih razloga povrede podataka u tradicionalnim okruženjima, ali i u okruženjima mikroservisa. Važno je razumjeti sigurnosne konfiguracije za svaku pojedinu komponentu Kubernetes klastera i činjenicu da svaka pogrešna sigurnosna postavka može otvoriti novu površinu za napad. Stoga je važno da sistemski administratori Kubernetes klastera razumiju različite sigurnosne prijetnje i pripadajuće konfiguracijske postavke.

U ovom dijelu specijalističkog rada biti će opisano kako osigurati sigurnost osnovnih komponenti Kubernetes klastera.

U mnogim slučajevima neće biti moguće slijediti sve najbolje sigurnosne prakse i preporuke, ali važno je istaknuti sigurnosne rizike i imati strategiju smanjivanja (kontroliranja, ublažavanja ili izbjegavanja) rizika, ako napadač pokuša iskoristiti određenu ranjivost u konfiguraciji Kubernetes klastera.

Sigurnost *Kube-apiserver*

Jedan od najvažnijih komponenti Kubernetes klastera prema kojemu treba obratiti najviše pažnje zasigurno jest *kube-apiserver.* Kube-apiserver je pristupnik ili prolaz (eng. Gateway) prema Kubernetes klasteru. To je središnja dodirna točka koja komunicira i upravlja sa gotovo pa svim ostalim komponentama unutar Kubernetes klastera. Kube-apiserver obavlja tri glavne funkcije, a to su:

1. **API menadžment:** *kube-apiserver* izlaže API-je za upravljanje klasterom. API-je koriste programeri, razvojni inženjeri i administratori klastera za izmjenu stanja klastera,
2. **Upravljanje zahtjevima:** Zahtjevi za upravljanje objektima i upravljanje klasterom potvrđuju se i obrađuju preko *kube-apiserver*-a,
3. **Interno slanje poruka:** API poslužitelj komunicira s drugim komponentama u klasteru kako bi osigurao ispravno funkcioniranje klastera, npr. sa *kublet* koji se nalazi na svim radnim čvorovima.

Prije same obrade, zahtjev prema API poslužitelju prolazi kroz sljedeće korake:

1. **Autentikacija:** *kube-apiserver* prvo provjerava izvor zahtjeva. kube-apiserver podržava više načina provjere autentičnosti uključujući klijentske certifikate, tokene i provjeru autentičnosti HTTP (eng. Hyper Text Transfer Protocol) zahtjeva
2. **Autorizacija:** Nakon što je identitet izvora provjeren, API poslužitelj potvrđuje da je izvoru dopušteno izvršiti zahtjev. *Kube-apiserver* prema zadanim postavkama podržava kontrolu pristupa temeljenu na atributima (ABAC), kontrolu pristupa temeljenu na ulogama (RBAC), autorizaciju čvorova i sl. RBAC je preporučeni način autorizacije koji je prethodno obrađen u ovom poglavlju.

3. **Kontrolor pristupa:** nakon što *kube-apiserver* provjeri autentičnost i autorizira zahtjev, kontroleri pristupa analiziraju zahtjev kako bi provjerili je li isti dopušten unutar Kubernetes klastera. Ako se zahtjev odbije, zahtjev se odbacuje.

Kube-apiserver je „mozak“ Kubernetes klastera. Kompromitacija API poslužitelja, gotovo u svim slučajevima uzrokuje kompromitaciju cijelog klastera, stoga je ključno da API poslužitelj bude siguran.

Sustav Kubernetes pruža mnoštvo postavki za konfiguriranje API poslužitelja. U daljnjem tekstu navesti će se sigurnosne konfiguracijske postavke koje trebaju biti postavljene u Kubernetes klasteru na ispravan način kako bi se osigurale osnovne sigurnosne pretpostavke za siguran i nesmetan rad klastera.

Slijede sigurnosne postavke Kube-apiserver-a:

- **Onemogućiti anonimnu provjeru autentičnosti:** Koristiti oznaku *anonymous-auth=false*. Ovo osigurava da se zahtjevi odbijeni od strane svih modula za provjeru autentičnosti ne tretiraju kao anonimni i da se odbacuju.
- **Onemogućiti osnovnu provjeru autentičnosti:** Osnovna provjera autentičnosti podržana je radi praktičnosti u *kube-apiserveru* i ne bi se trebala koristiti. Osnovne lozinke za provjeru autentičnosti traju neograničeno. *kube-apiserver* koristi argument *--basic-auth-file* za omogućavanje osnovne provjere autentičnosti. Potrebno je osigurati da se ovaj argument ne koristi.
- **Onemogućiti autentifikaciju tokenom:** *--token-auth-file* omogućuje na temelju tokena autentifikaciju za klaster. Ne preporučuje se provjera autentičnosti na temelju tokena. Statički tokeni traju zauvijek i potrebno je ponovno pokretanje API poslužitelja za ažuriranje tokena. Za autentifikaciju se trebaju koristiti klijentski certifikati.
- **Osigurati da veze s *kubelet-om* koriste HTTPS:** Prema zadanim postavkama, *--kubelet-https* postavljeno je na *true.* Osigurajte da ovaj argument nije postavljen na *false* za *kube-apiserver.*
- **Onemogućiti profiliranje:** Omogućavanje profiliranja pomoću *--profiling* otkriva nepotrebne pojedinosti o sustavu i programu. Osim ako se uoče problemi s performansama, onemogućiti profiliranje postavljanjem *--profiling=false*.
- **Onemogućiti *AlwaysAdmit:*** *--enable-admission-plugins* može se koristiti za omogućavanje dodataka za kontrolu pristupa koji nisu omogućeni prema zadanim postavkama. *AlwaysAdmit* prihvaća zahtjev.

- **Koristiti *AlwaysPullImages:*** kontrola pristupa *AlwaysPullImages* osigurava da se *Images* na čvorovima ne mogu koristiti bez točnih vjerodajnica. To sprječava zlonamjerne module da pokrenu spremnike za *Images* koje već postoje na čvoru.
- **Koristiti *SecurityContextDeny:*** Ovaj kontroler pristupa trebao bi se koristiti ako *PodSecurityPolicy* nije omogućen. *SecurityContextDeny* osigurava da Pod-ovi ne mogu modificirati *SecurityContext* za eskalaciju privilegija.
- **Omogućiti reviziju:** Revizija je omogućena prema zadanim postavkama u *kube-apiserver*. Provjeriti je li *--audit-log-path* postavljen na datoteku na sigurnoj lokaciji. Dodatno, osigurati da su parametri *maxage, maxsize i maxbackup* za reviziju (eng. Audit) postavljeni tako da ne predstavljaju probleme oko npr. prostora na disku i sl.
- **Onemogućiti *AlwaysAllow* autorizaciju:** način autorizacije osigurava da API poslužitelj analizira zahtjeve korisnika s ispravnim privilegijama. Ne koristiti *AlwaysAllow* s *--authorization-mode.*
- **Omogućiti RBAC autorizaciju:** RBAC je preporučeni način autorizacije za API poslužitelj. ABAC je težak za korištenje i upravljanje. Lakoća korištenja i jednostavnost ažuriranja RBAC *Roles* i *RoleBindings* čine RBAC prikladnim za okruženja koja se često skaliraju.
- **Osigurati da zahtjevi za kubelet koriste važeće certifikate:** Prema zadanim postavkama, *kube-apiserver* koristi HTTPS za zahtjeve za *kubelet.* Omogućavanje *--kubelet-certificateauthority, --kubelet-client-key* i *--kubelet-client-key* osigurava da komunikacija koristi važeće HTTPS certifikate.
- **Omogućiti traženje servisnog računa:** Osim osiguravanja da je token servisnog računa valjan, *kube-apiserver* također treba provjeriti da je token prisutan u *etcd.* Provjeriti da *--service-account-lookup* nije postavljen na vrijednost *false.*
- **Omogućiti *PodSecurityPolicy:*** mogu se koristiti *--enable-admission-plugins* kako bi omogućili *PodSecurityPolicy. PodSecurityPolicy* se koristi za definiranje sigurnosnih kriterija za Pod.
- **Koristiti ključnu datoteku servisnog računa:** Upotreba *--service-account-key-file* omogućuje rotaciju ključeva za račune usluge. Ako ovo nije navedeno, *kube-apiserver* koristi privatni ključ iz certifikata Transport Layer Security (TLS) za potpisivanje tokena računa usluge.

- **Omogućiti ovlaštene zahtjeve za etcd:** *--etcd-certfile* i *--etcd-keyfile* može se koristiti za identifikaciju zahtjeva. To osigurava da *etcd* može odbiti sve neidentificirane zahtjeve.
- **Nikako ne onemogućiti kontroler pristupa *ServiceAccount admission controller*:** Ova kontrola pristupa automatizira račune usluga. Omogućavanje ServiceAccount-a osigurava da se prilagođeni *ServiceAccount* s ograničenim dopuštenjima može koristiti s različitim Kubernetes objektima.
- **Ne koristiti samopotpisane certifikate:** ako je HTTPS omogućen za *kube-apiserver*, *--tls-cert-file* i *--tls-private-key-file* treba osigurati da se ne koriste samopotpisani certifikati (eng. Self-Signed Certificates)
- **Osigurati sigurne veze prema *etcd:*** Postavka *--etcd-cafile* omogućuje *kube apiserveru* da se potvrdi za *etcd* preko SSL-a (eng. Secure Sockets Layer) koristeći datoteku certifikata.
- **Koristiti sigurne TLS veze:** postavite *--tls-cipher-suites* samo na jake šifre. *--tls-min-version* koristi se za postavljanje najmanje podržane TLS verzije. Trenutno, TLS 1.2 preporučena je verzija TLS-a u sustavu Kubernetes.
- **Omogućiti naprednu reviziju sustava:** Napredna revizija se može onemogućiti postavljanjem *--feature-gates* na *AdvancedAuditing=false.* Svakako provjeriti ovu postavku.

Sigurnost *Kubelet*

Kubelet je agent koji se nalazi na svakom radnom čvoru (eng. Worker Nodes) u Kubernetes klasteru. *Kubelet* upravlja životnim ciklusom objekata unutar Kubernetes klastera i osigurava da su objekti u ispravnom stanju na čvoru gdje se nalaze.

Slijede sigurnosne postavke Kubelet-a:

- **Onemogućiti anonimnu provjeru autentičnosti:** ako je anonimna provjera autentičnosti omogućena, zahtjevi koji su odbijeni drugim metodama provjere autentičnosti tretiraju se kao anonimni. Osigurajte da je *--anonymous-auth=false* postavljeno za svaku instancu *kubelet.*
- **Postaviti način autorizacije:** Način autorizacije za *kubelet* postavlja se pomoću konfiguracijskih datoteka. Konfiguracijska datoteka navedena je pomoću parametra *--config.* Osigurati da način autorizacije nema *AlwaysAllow* na popisu.
- **Rotirati kubelet certifikate:** *kubelet* certifikate moguće je rotirati pomoću konfiguracije *RotateCertificates* u *kubelet* konfiguracijskoj datoteci. Ovo bi se trebalo koristiti u kombinaciji s *RotateKubeletServerCertificate* za automatsko traženje rotacije certifikata poslužitelja.
- **Onemogućiti priključak (eng. Port) samo za čitanje:** priključak samo za čitanje omogućen je za *kubelet* prema zadanim postavkama i trebao bi biti onemogućen. Port samo za čitanje poslužuje se bez provjere autentičnosti ili autorizacije!
- **Omogućiti kontroler pristupa *NodeRestriction:*** kontroler pristupa *NodeRestriction* dopušta *kubeletu* samo izmjenu čvora i Pod objekata na čvoru na koji je vezan.
- **Ograničiti pristup *Kubelet* API-ju:** Unutar Kubernetes klastera samo komponenta *kube-apiserver* komunicira s *kubelet API*. Ukoliko se pokuša komunicirati s *kubelet* API-jem na čvoru, to nije dozvoljeno. Ta komunikacija je osigurana samo korištenjem RBAC-a za *kubelet.*

Sigurnost „*etcd*“ baze podataka

Etcd je baza podataka koja pohranjuje ključeve i vrijednosti koju Kubernetes koristi za pohranu podataka. U *etcd* pohranjuju se stanja, konfiguracije i ostali podaci potrebni za normalno funkcioniranje Kubernetes klastera. Najbolja praksa govori da samo *kube-apiserver* treba imati pristup etcd-u. Kompromitiranje *etcd* može dovesti do kompromitiranja cijelog Kubernetes klastera!

Etcd pohranjuje sve podatke (pa i one najosjetljivije) Kubernetes klastera, kao što su privatni ključevi, certifikati, Secrets datoteke i sl.. Kompromitiranje *etcd* znači ujedno i kompromitiranje komponente api-poslužitelja.

Administratori Kubernetes klastera moraju obratiti posebnu pozornost prilikom postavljanja sigurnosnih postavki *etcd* baze podataka i to odmah prilikom kreiranja tj. pokretanja Kubernetes klastera.

Slijede sigurnosne postavke etcd baze podataka:

- **Ograničiti pristup čvoru:** Koristiti Linux vatrozid kako biste osigurali da samo čvorovi koji trebaju pristup *etcd*-u imaju pristup.
- **Osigurati da API poslužitelj koristi TLS:** *--cert-file* i *--key-file.* Na taj način osiguravamo da zahtjevi prema *etcd* bazi podataka budu sigurni.
- **Koristiti valjane certifikate:** *--client-cert-auth* osigurava da se komunikacija s klijentima vrši pomoću valjanih certifikata, a postavljanje *--auto-tls* na *false* osigurava da se samopotpisani certifikati (eng. Self-Signed Certificates) ne koriste.
- **Šifrirati podatke u mirovanju:** *--encryption-provider-config* prosljeđuje se API-ima kako bi se osiguralo da su podaci šifrirani dok miruju u *etcd* bazi podataka.

Sigurnost *Kube-scheduler*

Kube-scheduler je odgovoran za dodjeljivanje čvora određenom Pod-u. Drugim riječima, prilikom kreiranja Pod-a, *Kube-scheduler* određuje (na temelju određenih parametara i stanja Kubernetes radnih čvorova) na koji će se čvor postaviti Pod.

Nakon što je Pod dodijeljen čvoru, *kubelet* izvršava Pod. *Kube-scheduler* prvo filtrira skup čvorova na kojima se Pod može pokrenuti, a zatim, na temelju bodovanja svakog čvora, dodjeljuje Pod filtriranom čvoru s najvećim rezultatom. To je proces koji se odvija određenom automatikom unutar Kubernetes klastera i to se u ovom specijalističkom radu neće posebno obrađivati.

Kompromitiranje komponente *kube-scheduler* utječe na performanse i dostupnost Pod-ova u Kubernetes klasteru, a to samo po sebi znači da, u određenim slučajevima, kada je narušena sigurnost *kube-scheduler* može doći i do nedostupnosti Pod-ova pa tako i spremnika te na kraju i npr. aplikacija koji se nalaze unutar istih.

Slijede sigurnosne postavke *kube-scheduler*-a:

- **Onemogućiti profiliranje:** Profiliranje *kube-schedulera* otkriva detalje sustava. Postavljanje *--profiling* na *false* smanjuje površinu napada (eng. Attack Surface).
- **Onemogućiti vanjske veze s *kube-schedulerom*:** vanjske veze trebaju biti onemogućene za *kube-scheduler*. *AllowExtTrafficLocalEndpoints* treba biti postavljen na *true*, te na taj način onemogućiti vanjske veze s *kube-schedulerom*. Svakako provjeriti da li je ova značajka onemogućena pomoću *--feature-gates.*
- **Omogućiti *AppArmor*:** Prema zadanim postavkama, *AppArmor* je omogućen za *kube-scheduler.* Svakako dodatno provjeriti da *AppArmor* nije onemogućen za *kube-scheduler.*

Sigurnost *Kube-controller-manager*

Kube-controller-manager upravlja kontrolnom petljom u Kubernetes klasteru. On prati promjene klastera putem API poslužitelja i ima za cilj promijeniti klaster iz trenutnog stanja u željeno stanje (eng. Current State to Desired State) kao jednog od osnovnih postulata sustava Kubernetes.

Višestruki upravitelji kontrolera isporučuju se prema zadanim postavkama s *kube-controller*-managerom, kao što je *Replication Controller* i *Namespace Controller.*

Ugrožavanje *kube-controller-manager*-a može rezultirati da ažuriranje klastera bude odbijeno tj. da se ne odrađuje ili da se ne odrađuje kako je uobičajeno.

Kako bi se osigurao *kube-controller-manager*, potrebno je upotrijebiti *--use-serviceaccount-credentials* koje, kada se koriste s RBAC-om, osiguravaju pokretanje kontrolnih petlji s minimalnim privilegijama, a što je u skladu sa osnovnim sigurnosnim preporukama za sustav Kubernetes.

Sigurnost Kubernetes *CoreDNS*

Kube-dns bio je zadani DNS (eng. Domain Name System) poslužitelj za Kubernetes klaster.

DNS poslužitelj pomaže internim objektima kao što su *Services,* Pod-ovi i spremnici da lociraju jedni druge „via“ DNS.

Kube-dns se sastoji od tri spremnika (eng. Container):

- ***kube-dns:*** Ovaj spremnik koristi SkyDNS za izvođenje usluga razlučivanja (eng. Resolution) DNS-a.
- ***dnsmasq:*** lagani (eng. Lightweight) DNS razrješivač (eng. Resolver) koji predmemorira (eng. Caches) odgovore SkyDNS-a.
- ***sidecar:*** nadzire stanje i obrađuje izvještaje za DNS. *kube-dns* je zamijenio *CoreDNS* od verzije 1.11 zbog sigurnosnih propusta u *dnsmasq* i problema s performansama u SkyDNS-u.

CoreDNS danas (u novijim verzijama Kubernetes sustava) je jedan spremnik koji pruža sve funkcije kube-dns-a.

Slijede sigurnosne postavke *CoreDNS*-a:

- **Provjeriti da „*health plugin*“ nije onemogućen:** „*health plugin*“ nadzire status CoreDNS-a. Koristi se za potvrdu radi li CoreDNS. Omogućuje se dodavanjem stanja na popis dodataka koji će se omogućiti u *Corefile*.
- **Omogućiti Istio dodatak (eng. Plugin) za CoreDNS:** Istio je servisna mreža koju koristi sustav Kubernetes za pružanje otkrivanja usluge (eng. Service Discovery), balansiranja opterećenja i autentifikacije. Nije dostupan prema zadanim postavkama u sustavu Kubernetes te ga je potrebno posebno postaviti, konfigurirati zajedno sa svim vanjskim ovisnostima (eng. External Dependency).

3.5 Sigurnost kontejnera i Pod-ova u sustavu Kubernetes

Pod-ovi su najmanje jedinice koje možemo izraditi i kojima možemo upravljati u sustavu Kubernetes.

Pod je grupa od jednog ili više spremnika (eng. Container), sa zajedničkim resursima za pohranu (eng. Storage) i mrežu (eng. Network), te specifikacijom za pokretanje spremnika. Sadržaj Pod-a uvijek je lociran i raspoređen te se izvodi u zajedničkom kontekstu.

Pod modelira tzv. "logički host" i specifičan je za pojedinu aplikaciju. Pod može sadržavati jedan ili više spremnika koji su u toj situaciji vrlo čvrsto povezani.

U sustavima koji nisu u oblaku (eng. Cloud), aplikacije koje se izvode na istom fizičkom ili virtualnom računalu analogne su aplikacijama u oblaku koje se izvode na istom logičkom hostu.

Pod-ovi u Kubernetes klasteru koriste se na dva glavna načina:

1. **Pod-ovi koji pokreću jedan spremnik.** Model "jedan spremnik - jedan pod" je najčešći slučaj korištenja Pod-ova u sustavu Kubernetes. U ovom slučaju, Pod možemo zamisliti kao omot oko jednog spremnika. Sustav Kubernetes upravlja Pod-ovima umjesto da izravno upravlja spremnicima.
2. **Pod-ovi koji pokreću više spremnika koji moraju raditi zajedno.** Pod može enkapsulirati aplikaciju sastavljenu od više zajedno smještenih spremnika koji su čvrsto povezani i trebaju dijeliti resurse. Zajedno smješteni spremnici tvore jedinstvenu kohezivnu jedinicu usluge. Na primjer, jedan spremnik izlaže podatke pohranjene u zajedničkom volumenu prema internetu, dok drugi, odvojeni spremnik osvježava ili ažurira te podatke i/ili datoteke. Pod objedinjuje ove spremnike, resurse za pohranu i mrežni identitet kao jednu jedinicu.

Iako je Pod najmanja jedinica koja služi kao mjesto za pokretanje mikroservisa tj. spremnika, sigurnost Kubernetes Pod-ova je vrlo opsežna tema koja pokriva cijeli tok DevOps procesa kao što je izgradnja, implementacija i vrijeme izvođenja (eng. Build, Deployment, and Runtime) stoga će se ova tema u ovom specijalističkom radu obraditi samo na osnovnoj sigurnosnoj razini.

Prevedeno s engleskog jezika sinonim „**DevOps**“ podrazumijeva skup praksi koje kombiniraju razvoj softvera i IT operacije. Cilj mu je skratiti životni ciklus razvoja sustava i aplikacija te osigurati kontinuiranu isporuku uz visoku kvalitetu i sigurnost softvera.

DevOps je komplementaran agilnom razvoju softvera tj. nekoliko DevOps aspekata proizašlo je iz agilnog načina rada.

U ovom specijalističkom radu fokus je na fazama izgradnje i izvođenja Pod-ova. Kako bismo osigurali Kubernetes Pod-ove u fazi izgradnje, potrebno je uspostaviti sigurnost spremnika i konfigurirati sigurnosne atribute Pod-ova (ili predložaka Pod-ova) kako bismo smanjili površinu za napad (eng. Attack Surface).

Isto tako, moramo biti svjesni i činjenice da iskorištavanje ranjivosti same aplikacije koja je smještena u spremniku pojedinog Pod-a može dovesti i do kompromitiranja Pod-a pa i cijelog sustava Kubernetes ili Kubernetes klastera.

Međutim, i to spada u DevOps dio tj. dio koji se odnosi na projektiranje, razvoj i izradu sigurnih aplikacijskih rješenja te tema nije dio ovog specijalističkog rada.

Konfiguriranje sigurnosnih atributa Pod-ova

Kao što je već spomenuto, programeri aplikacija trebaju biti svjesni koje privilegije mikroservis i/ili aplikacija mora imati da bi mogao izvršavati zadatke. U idealnom slučaju, programeri aplikacija, sigurnosni inženjeri i administratori sustava Kubernetes rade zajedno na ojačavanju mikroservisa na razini Pod-a i spremnika konfiguriranjem sigurnosnog konteksta koji pruža sustav Kubernetes.

Dakle, svi uključeni moraju biti upućeni u određene sigurnosne aspekte koji se moraju zadovoljiti kako bi Pod, aplikacija, pripadajući mikroservis i konteiner bio siguran.

Na razini Pod-a postoje glavni sigurnosni atributi:

- Postavljanje *NameSpace* za Pod-ove na nivou hosta
- Sigurnosni kontekst na razini spremnika
- Sigurnosni kontekst na razini Pod-a
- Konfiguracija *PodSecurityPolicy* (nije dostupan od verzije 1.25 sustava Kubernetes!)

Postavljanje *NameSpaces* za Pod-ove na nivou hosta

U specifikaciji modula koriste se sljedeći atributi za konfiguriranje *NameSpace* na nivou hosta:

- ***hostPID:*** Prema zadanim postavkama, ovo je postavljeno na *false*. Postavljanje na *true* omogućuje modulu vidljivost svih procesa u radnom čvoru.
- ***hostNetwork:*** Prema zadanim postavkama, ovo je postavljeno na *false*. Postavljanje na *true* omogućuje modulu vidljivost na svim mrežnim nivoima na radnom čvoru.
- ***hostIPC:*** Prema zadanim postavkama, ovo je postavljeno na *false*. Postavljanje na *true* omogućuje Podu da ima vidljivost na svim IPC resursima u radnom čvoru.

Sigurnosni kontekst za spremnike (eng. Containers)

Kao što je opisano u uvodu ovog poglavlja više spremnika može se grupirati zajedno unutar istog Pod-a. Svaki spremnik može imati vlastiti sigurnosni kontekst, koji definira privilegije i kontrole pristupa. Dizajn sigurnosnog konteksta na razini spremnika pruža detaljniju sigurnosnu kontrolu.

Na primjer, možemo imati tri spremnika koji rade unutar istog Pod-a. Jedan od njih mora raditi u povlaštenom načinu rada, dok ostali rade u nepovlaštenom načinu rada. To se može konfigurirati na način da se postavi sigurnosni konteksta za svaki pojedinačni spremnik.

Slijede glavni atributi sigurnosnog konteksta za spremnike:

- ***privileged*:** Prema zadanim postavkama, ovo je postavljeno na *false.* Postavljanje na *true* u biti čini procese unutar spremnika ekvivalentnima „root“ korisniku na radnom čvoru.
- ***capabilities*:** postoji zadani skup mogućnosti koje je spremniku dodijelilo vrijeme izvođenja spremnika.

  Zadane dodijeljene mogućnosti su sljedeće: *CAP_SETPCAP, CAP_MKNOD, CAP_AUDIT_WRITE, CAP_CHOWN, CAP_NET_RAW, CAP_DAC_OVERRIDE, CAP_SETGID, CAP_NET_BIND_SERVICE, CAP_FOWNER, CAP_FSETID, CAP_KILL, CAP_SETUID, CAP_SYS_CHROOT i CAP_FC.*

  Konfiguriranjem ovih atributa može se dodati dodatne mogućnosti ili izbaciti neke od zadanih vrijednosti. *CAP_SYS_ADMIN* i *CAP_NETWORK_ADMIN* treba dodavati s oprezom. Što se tiče zadanih mogućnosti, treba odbaciti sve one koje su nepotrebne.

- ***allowPrivilegeEscalation:*** Prema zadanim postavkama ovo je postavljeno na *true.* Njegovo postavljanje izravno kontrolira *no_new_privs* koja će biti postavljena na procese u spremniku. U osnovi, ovaj atribut kontrolira može li proces dobiti više privilegija od svog nadređenog procesa. Treba imati na umu da ako spremnik radi u privilegiranom načinu rada ili mu je dodana mogućnost *CAP_SYS_ADMN*, ovaj će se atribut automatski postaviti na *true.* Dobra praksa govori da je ovu vrijednost potrebno postaviti na *false*.
- ***readOnlyrootFilesystem:*** Prema zadanim postavkama, ovo je postavljeno na *false.* Postavljanje na *true* čini korijenski datotečni sustav spremnika samo za čitanje, što znači da su to datoteke biblioteke, konfiguracijske datoteke i sl.
- ***runAsNonroot:*** Prema zadanim postavkama, ovo je postavljeno na *false.* Postavljanje na *true* omogućuje provjeru valjanosti da se procesi u spremniku ne mogu izvoditi kao „root“ korisnik (UID=0). Provjeru valjanosti vrši *kubelet.*

  S *runAsNonroot* postavljenim na *true, kubelet* će spriječiti pokretanje spremnika ako se pokrene kao „root“ korisnik. Dobra je sigurnosna praksa postaviti ga na *true.* Ovaj je atribut također dostupan u *PodSecurityContext*, koji stupa na snagu na razini modula. Ako je ovaj atribut postavljen i u *SecurityContext* i u *PodSecurityContext,* vrijednost navedena na razini spremnika ima prednost.
- ***runAsUser:*** Ovo je dizajnirano za određivanje UID-a za pokretanje procesa ulazne točke slike spremnika. Zadana postavka je korisnik naveden u metapodacima Image-a (na primjer, *USER* u Docker datoteci). Ovaj je atribut također dostupan u *PodSecurityContext*, koji stupa na snagu na razini modula. Ako je ovaj atribut postavljen i u *SecurityContext* i u *PodSecurityContext*, vrijednost navedena na razini spremnika ima prednost.
- ***runAsGroup:*** Slično kao i *runAsUser*, ovo je dizajnirano za određivanje ID-a grupe ili GID-a za pokretanje procesa ulazne točke spremnika. Ovaj je atribut također dostupan u *PodSecurityContext*, koji stupa na snagu na razini modula. Ako je ovaj atribut postavljen i u *SecurityContext* i u *PodSecurityContext,* vrijednost navedena na razini spremnika ima prednost.
- ***seLinuxOptions:*** Ovo je dizajnirano za specificiranje SELinux konteksta za pojedini spremnik. Prema zadanim postavkama, *runtime* spremnika dodijelit će nasumični SELinux kontekst spremniku ako nije naveden. Ovaj je atribut također dostupan u *PodSecurityContexu*, koji stupa na snagu na razini modula.

Slijedom svega navedenog, postavke sigurnosnih atributa i cijelog sigurnosnog konteksta, svakako je potrebno (i uputno) uskladiti s poslovnim zahtjevima i karakteristikama pojedinih mikroservisa i/ili aplikativnih rješenja.

Općenito govoreći, najbolja sigurnosna praksa je sljedeća:

- Ne pokretati spremnike u privilegiranom načinu rada osim ako je to neizbježno,
- Ne dodavati dodatne mogućnosti osim ako nije prijeko potrebno,
- Izostaviti neiskorištene zadane mogućnosti / atribute,
- Pokretati spremnike kao *non-root* korisnik,
- Omogućiti provjeru *runAsNonroot* atributa,
- Postaviti „root“ datotečni sustav spremnika samo kao RO (eng. Read-Only).

Sigurnosni kontekst na razini Pod-a

Na samom početku vrlo važno je napomenuti da će se sigurnosni kontekst koji se primjenjuje na razini Pod-a, primijeniti na sve spremnike unutar Pod-a. Drugim riječima, to ujedno znači da će se primijenjeni sigurnosni atributi primijeniti na sve spremnike unutar Pod-a.

Slijede glavni atributi sigurnosnog konteksta za Pod-ove:

- ***fsGroup:*** Ovo je posebna dopunska grupa primijenjena na sve spremnike. Učinkovitost ovog atributa ovisi o vrsti volumena. U biti, omogućuje *kubelet-u* da postavi vlasništvo nad montiranim volumenom na Pod s dodatnim GID-om.
- ***sysctls:*** *sysctls* se koristi za konfiguriranje parametara jezgre tijekom izvođenja. U takvom kontekstu, *sysctls* i kernel parametri se koriste naizmjenično. Ove *sysctls* naredbe su parametri kernela s *Namesopace* koji se primjenjuju na Pod. Poznato je da su sljedeće sysctls naredbe imenskog prostora: kernel.shm*, kernel. msg*, kernel.sem i kernel.mqueue.*. Nesiguran *sysctl* onemogućen je prema zadanim postavkama i ne bi trebao biti omogućen u produkcijskim okruženjima.
- ***runAsUser:*** Ovo je dizajnirano za određivanje UID-a za pokretanje procesa ulazne točke slike spremnika. Zadana postavka je korisnik naveden u metapodacima Image-a (na primjer, uputa USER u datoteci Docker). Ovaj je atribut također dostupan u *SecurityContextu,* koji stupa na snagu na razini spremnika. Ako je ovaj atribut postavljen iu *SecurityContext* iu *PodSecurityContext,* vrijednost navedena na razini spremnika ima prednost.

- ***runAsGroup:*** Slično kao i *runAsUser,* ovo je dizajnirano za određivanje GID-a za pokretanje procesa ulazne točke spremnika. Ovaj je atribut također dostupan u SecurityContextu, koji stupa na snagu na razini spremnika. Ako je ovaj atribut postavljen i u *SecurityContext* i *PodSecurityContext,* vrijednost navedena na razini spremnika ima prednost.
- ***runAsNonRoot:*** Prema zadanim postavkama postavljena na *false*. Postavljanje na *true* omogućuje provjeru valjanosti da se procesi u spremniku ne mogu izvoditi kao „root“ korisnik (UID=0). Provjeru valjanosti vrši kubelet. Postavljanjem na true, kubelet će spriječiti pokretanje spremnika ako se pokrene kao „root“ korisnik. Dobra je sigurnosna praksa postaviti ga na *true.* Ovaj je atribut također dostupan u *SecurityContextu,* koji stupa na snagu na razini spremnika. Ako je ovaj atribut postavljen i u *SecurityContext* i u *PodSecurityContext,* vrijednost navedena na razini spremnika ima prednost.
- ***seLinuxOptions:*** Ovo je dizajnirano za specificiranje SELinux konteksta za spremnik. Prema zadanim postavkama, runtime spremnika dodijelit će nasumični SELinux kontekst spremniku ako nije naveden. Ovaj je atribut također dostupan u *SecurityContextu,* koji stupa na snagu na razini spremnika. Ako je ovaj atribut postavljen i u *SecurityContext* i u *PodSecurityContext,* vrijednost navedena na razini spremnika ima prednost.

PodSecurityPolicy

PodSecurityPolicy smatra se „zastarjelim“ od verzije 1.21. Sa verzijom sustava Kubernetes 1.25 više nije dostupan i kao takav se ne može koristiti. No, pošto je u ovom trenutku velika većina Kubernetes klastera postavljena na verzijama ispod 1.25 ova tema je još uvijek vrlo aktualna i vrijedna spomena.

Od Ver. 1.25 umjesto *PodSecurityPolicy* u upotrebi je *Pod Security Admission* standard tj. sigurnosni kontekst koji će se zasebno opisati u nastavku ovog poglavlja.

Kubernetes *PodSecurityPolicy* je resurs na razini Kubernetes klastera koji kontrolira sigurnosno osjetljive aspekte specifikacije Pod-ova putem kojih se ograničavaju privilegije za pristup Kubernetes Pod-ovima.

Za kreiranja *PodSecurityPolicy* može se koristiti alat otvorenog koda, „*kube-psp-advisor*“, koji može pomoći u izgradnji prilagodljive sigurnosne politike za Pod-ove na razini cijelog Kubernetes klastera.

PodSecurityPolicy možemo zamisliti kao sigurnosnu politiku za procjenu sigurnosnih atributa definiranih u specifikaciji Pod-a. Cilj je da samo oni Pod-ovi čiji sigurnosni atributi ispunjavaju sve zahtjeve navedene u *PodSecurityPolicy* biti će pridodani u Kubernetes klaster.

Kao jednostavan primjer *PodSecurityPolicy* može se koristiti za blokiranje pokretanja većine privilegiranih Pod-ova, dok se samo onim potrebnim ili ograničenim Pod-ovima dopušta pristup datotečnom sustavu hosta.

Slijede glavni atributi sigurnosnog konteksta za *PodSecurityPolicy*:

- ***privileged:*** Određuje može li se modul pokretati u povlaštenom načinu rada.
- ***hostPID:*** Određuje može li Pod koristiti *NameSpace* hosta.
- ***hostNetwork:*** Određuje može li Pod koristiti prostor imena host mreže.
- ***hostIPC:*** Određuje može li Pod koristiti imenski prostor IPC hosta. Zadana postavka je *true.*
- ***allowedCapabilities:*** Određuje popis mogućnosti koje se mogu dodati spremnicima. Zadana postavka je prazna.
- ***defaultAddCapabilities:*** Određuje popis mogućnosti koje će biti dodane spremnicima prema zadanim postavkama. Zadana postavka je prazna.

- ***requiredDropCapabilities:*** Određuje popis mogućnosti koje će biti ispuštene iz spremnika. Moramo znati da se sposobnost ne može navesti u oba polja *allowedCapabilities* i *requiredDropCapabilities*. Zadana postavka je prazna.
- ***readOnlyRootFilesystem:*** Kada je postavljeno na *true, PodSecurityPolicy* će prisiliti spremnike da rade s korijenskim datotečnim sustavom samo za čitanje. Ako je atribut izričito postavljen na *false* u sigurnosnom kontekstu spremnika, bit će odbijeno pokretanje modula. Zadana postavka je *false.*
- ***runAsUser:*** Određuje dopuštene ID-ove korisnika koji se mogu postaviti u sigurnosnom kontekstu Pod-a i spremnika. Zadana postavka dopušta sve.
- ***runAsGroup:*** Određuje dopuštene grupne ID-ove koji se mogu postaviti u sigurnosnom kontekstu Pod-a i spremnika. Zadana postavka dopušta sve.
- ***allowPrivilegeEscalation:*** Određuje može li grupa podnijeti zahtjev za dopuštanje eskalacije privilegija. Zadana postavka je *true.*
- ***allowedHostPaths:*** Određuje popis staza hosta koje bi modul mogao montirati. Zadana postavka dopušta sve.
- ***volumes:*** Određuje popis tipova volumena koje može montirati (eng. Mount) Pod. Na primjer, secret, configmap i hostpath su važeće vrste volumena. Zadana postavka dopušta sve.
- ***seLinux:*** Određuje dopuštene SElinux oznake koje se mogu postaviti u sigurnosnom kontekstu Pod-ova i spremnika. Zadana postavka dopušta sve.
- ***allowedUnsafeSysctl:*** Omogućuje pokretanje nesigurnih sysctl-a. Zadana postavka ne dopušta ništa.

Jednostavan primjer *PodSecurityPolicy*:

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: pod-security-policy-psp
spec:
  privileged: false  # Ne ukljucuje privilegirane Pod-ove!
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  volumes:
    - '*'
```

Pod Security Admission

Kao što je već rečeno, *PodSecurityPolicy* smatra se „zastarjelim“ od **Ver. 1.21** sustava Kubernetes. Ali, tek sa verzijom sustava Kubernetes **Ver. 1.25** više nije dostupan i kao takav se ne može koristiti.

Od **Ver. 1.25** umjesto *PodSecurityPolicy* u upotrebi je *Pod Security Admission* standard tj. sigurnosni kontekst i to u stabilnoj verziji.

Isto tako, u kontekstu nadogradnje sustava Kubernetes na Ver. 1.25. i više, valja napomenuti da je prelazak na produkcijskim okolinama sustava Kubernetes sa *PodSecurityPolicy na Pod Security Admission* veoma kompliciran i zahtjevan proces koji iziskuje vrlo detaljnu pripremu, kao i detaljno i studiozno testiranje prije same implementacije. Pogotovo ukoliko se radi o sustavima sa velikim brojem čvorova, Pod-ova, implementacija, aplikacija i sl.

Kubernetes nudi ugrađeni kontroler pristupa za sigurnost Pod-ova tj. *Pod Security Admission Controller* koji je odgovoran za provođenje Pod sigurnosnih standarda. Sigurnosna ograničenja Pod-ova primjenjuju se na razini *Namespace* o kojima se podovi kreiraju.

Pod Security Admission* oznake (eng. Labels) na nivou *Namespaces

Nakon što je *Pod Security Admission* omogućen mogu se kreirati i konfigurirati *Namespaces* kako bi definirali način kontrole pristupa koji želimo koristiti za sigurnost Pod-ova u svakom pojedinačnom *Namespace-u.*

Kubernetes definira skup oznaka koje možemo postaviti kako bi smo definirali koju od unaprijed definiranih razina sigurnosnog standarda Pod-a želimo koristiti za određeni *Namespace.* Oznaka koju odaberemo definira koju radnju poduzima kontrolna razina ako se otkrije potencijalno kršenje:

(1) *Enforce* - Kršenje pravila uzrokovat će odbijanje Pod-a

(2) *Audit* - Kršenja pravila pokrenut će dodavanje revizijske zabilješke događaju zabilježenom u revizijskom dnevniku, ali su inače dopuštena.

(3) *Warn* - Kršenja pravila pokrenut će upozorenje upućeno korisniku, ali su inače dopuštena.

Razine sigurnosti na nivou Pod-a (eng. Pod Security Levels)

Sigurnost Pod-a postavlja zahtjeve na cjelokupan sigurnosni kontekst Poda, ali i na druga srodna polja u skladu s tri (3) razine definirane sigurnosnim standardima na nivou Pod-a: **(1)** *Privileged,* **(2)** *Baseline, i* **(3)** *Restricted*.

Pod sigurnosni standardi (eng. Pod Security Standards)

Sigurnosni standardi Pod-a definiraju tri različite politike kako bi što šire pokrili sigurnosni spektar. Ta su pravila kumulativna i kreću se od vrlo permisivnih do visoko restriktivnih. Slijedi kratki opis zahtjeve svake politike:

(1) ***Privileged*** **-** Neograničena politika, koja pruža najširu moguću razinu dopuštenja. Ovo pravilo dopušta poznate eskalacije privilegija.

(2) ***Baseline*** - Minimalno restriktivna politika koja sprječava poznate eskalacije privilegija. Omogućuje zadanu (minimalno specificiranu) konfiguraciju Poda.

(3) ***Restricted*** - Strogo ograničena politika, slijedeći trenutne najbolje sigurnosne prakse

Uza sve navedeno vrlo je važno napomenuti da možemo primijeniti više od jednog načina na određeni *Namespace.* Ovo je korisno za provođenje jednog standarda politike, a upozoravanje i reviziju drugog.

U donjem primjeru prikazano je kako Pod-ovi u *Namespace* pod nazivom „psa-test“ koji krše *Baseline* politiku neće biti dopušteni (Enforce), a Pod-ovi koji krše *Restricted* politiku ući će u *Audit* i *Warn.*

```
apiVersion: v1
kind: Namespace
metadata:
  name: psa-test
  labels:
    pod-security.kubernetes.io/enforce: baseline
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted
```

3.6 Sigurnost *Secret* objekata

Gotovo pa svaka aplikacija ili programsko rješenje sadrži neke podatke koji moraju biti tajni, a u većini slučajeva to su pristupni podatci za pristup sustavu, bazi podataka ili nekom drugom objektu i sl..

Tajni podaci mogu biti u obliku od vjerodajnica (korisničko ime i lozinka) pa do TLS certifikata ili pristupnih tokena za uspostavljanje sigurnih veza.

Razvojni tim, kada izrađuju aplikacijsko rješenje trebao bi osigurati siguran način upravljanja takvim tajnim podacima. Zbog toga sustav Kubernetes sadrži objekt pod nazivom ***Secret*** koji služi upravo tome tj. da pohranjuje osjetljive podataka koji bi inače mogli biti ugrađeni kao dio u specifikaciji Pod-a ili slike (eng. Image) spremnika aplikacije, što nikako nije dobra praksa.

Slijede osnovne pojedinosti o *Secret* objektu unutar sustava Kubernetes:

- *Secret* se stvara izvan Pod-ova - *Secret* se stvara prije no što se bilo koji Pod može koristiti.
- Kada se stvori *Secret* objekt, on se pohranjuje unutar Kubernetes pohrane podataka tj. *etcd* baze podataka u Kubernetes *Control Plane* nivou tj. na glavnom, *Master* čvoru,
- Prilikom kreiranja *Secret* objekta, specificira se *data* i/ili *stringData* polja. Vrijednosti za sva polja podataka moraju biti base64-kodirani niz,
- Prilikom kreiranja *Secret*, ograničenje je na veličinu od 1 MB. Ovo ograničenje je postavljeno kako bi se onemogućilo stvaranje velikih datoteka koje bi onda iste mogle opteretiti *kube-apiserver* i *kubelet* memoriju,
- Prilikom kreiranja *Secret*, može ih se označiti kao nepromjenjive s atributom *immutable:true.* kako bi se spriječile izmjene podataka nakon izrade *Secret* datoteke.
  Isto tako, označavanje *Secret* kao nepromjenjive štiti od slučajnih ili neželjenih ažuriranja/izmjena koja bi mogla uzrokovati prekid rada aplikacije,
- Nakon što se *Secret* kreira, ubacuj ga se u Pod bilo montiranjem (eng. Mounting) kao volumen podataka (eng. Data Volumes), izlaganjem kao varijable okruženja (eng. Environment variables), ili kao *imagePullSecrets.*

Vrste *Secret* objekata sustava Kubernetes

Opaque Secrets: koriste se za pohranjivanje proizvoljnih korisnički definiranih podataka. *Opaque Secrets* je vrsta *Secret*, što znači da će se *Secret* smatrati kao *Opaque Secrets* kada se kreira *Secret* i ne navede se vrsta,

Service account token Secrets: ova vrsta koristi se za pohranu vjerodajnice tokena koja identificira servisni račun (eng. Service Account). Važno je napomenuti da kada se koristi ova vrsta, mora se osigurati da je napomena *kubernetes.io/service-account.name* postavljena na postojeći naziv servisnog računa,

Docker config Secrets: koristi se za konfiguracije Dockera za pohranjivanje vjerodajnica za pristup registru slika spremnika. Koristiti *Docker config secret* s jednom od sljedećih vrijednosti tipa: *kubernetes.io/dockercfg* i *kubernetes.io/dockerconfigjson,*

Basic authentication Secret: ova vrsta koristi se za pohranjivanje vjerodajnica potrebnih za osnovnu autentifikaciju. Kada se koristi *Basic authentication Secret*, podatkovno polje mora sadržavati barem jedan od sljedećih ključeva: korisničko ime (korisničko ime za provjeru autentičnosti) i lozinka (lozinka ili token za autentifikaciju),

SSH authentication secrets: ova vrsta koristi se za pohranu podataka koji se koriste u SSH autentifikaciji. Kada se koristi SSH autentifikacija, mora se navesti *ssh-privatekey key-value pair* u polju podataka (ili *stringData*) kao SSH vjerodajnicu za korištenje,

TLS secrets: ova vrsta koristi se za pohranjivanje certifikata i pridruženog ključa koji se obično koristi za TLS. Kada se koristi *TLS Secret*, mora se unijeti *tls.key* i *tls.crt* ključ u konfiguracijskim podacima (ili *stringData*),

Bootstrap token Secrets: ova vrsta koristi se za pohranu podataka tokena za pokretanje tijekom procesa pokretanja Kubernetes čvora. Obično se kreira *bootstrap token Secret* u *kube-system Namespace* i imenuje ga se u obliku *bootstrap-token-<token-id>*.

Prije no što se kreira Secret pomoću manifest datoteke, mora se odlučiti kako se želi dodati *Secret* podatke pomoću podatkovnog polja *data* ili *stringData* polja. Koristeći podatkovno polje *data*, tajni podatci kodiraju se koristeći *base64*. Da bi se *username* i *password* pretvorio u *base64*, potrebno je pokrenuti sljedeće naredbe:

echo -n 'admin' | base64

echo -n 'password' | base64

Primjer Secret datoteke *(secret-primjer.yaml)* **kao manifest datoteke koristeći *data* polje:**

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-primjer
type: Opaque
data:
  username: YWRtaW4=
  password: cGFzc3dvcmQ=
```

Za kreiranje Secret objekta u *Namespace* ***secret-demo-namespace*** u sustavu Kubernetes u ovom primjeru koristimo slijedeću naredbu:

```
# kubectl create namespace secrets-demo-namespace
# kubectl -n secret-demo-namespace apply -f secret-primjer.yaml
```

Ista *Secret* datoteka koristeći *stringData* polje sadržavala bi:

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-primjer
type: Opaque
stringData:
  username: admin
  password: password
```

Kubernetes *Secrets* se prema zadanim postavkama pohranjuju nekriptirane u *etcd* bazu podataka. Svatko s osiguranim pristupom API-ju može dohvatiti ili izmijeniti Secret datoteku kao i svatko s pristupom *etcd* bazi podataka.

Osim toga, svatko tko je ovlašten stvoriti Pod u određenom *Namespace*-u može koristiti taj pristup za čitanje bilo koje *Secret* datoteke u tom *Namespace*-u. Ovo uključuje i neizravan pristup kao što je mogućnost stvaranja implementacije tj. *Deployment* u sustavu Kubernetes.

Kako bi se sigurno koristili *Secrets* objekti, moraju se poduzeti neki od slijedećih koraka:

- Omogućiti enkripciju za *Secret* datoteke u mirovanju,
- Omogućiti RBAC pravila s najmanjim privilegiranim pristupom *Secret* datotekama,
- Ograničiti pristup od strane *Secret* datoteka određenim spremnicima,
- Razmisliti o korištenju vanjskih pružatelja usluga kreiranja, manipulacije i sl. *Secret* objektima u sustavu Kubernetes kao što je **Vault.** Više o samom sustavu može se vidjeti na slijedećem linku: (https://www.hashicorp.com/products/vault)

Kako dodatno zaštititi *Secret* objekte

Uobičajen pristup za sigurnije upravljanje podacima unutar *Secret* objekata u sustavu Kubernetes je uvođenje vanjskog rješenja za upravljanje tajnim podacima, kao što su npr. Hashicorp Vault, AWS Secrets Manager, Azure Key Vault ili Google Secret Manager.

Dok ovaj pristup rješava neke probleme svojstvene sustavu Kubernetes, drugi, općenitiji izazovi upravljanja *Secret* objektima i dalje postoje. Upotrebom rješenja „treće strane“ još uvijek je gotovo pa nemoguće osigurati da:

- *Da Secret* objekti budu dostupni samo onim spremnicima koji ih stvarno trebaju. Čvrsta integracija s sustavom Kubernetes potrebna je za mapiranje *Secret* objekata u spremnike i za dobivanje odgovarajućeg uvida u to,
- Da se Secret objekti ne pohranjuju se na disk ili u obliku varijabli okoline (eng. Environment Variables),
- Da aplikacije ne mogu odavati podatke unutar *Secrets* objekata,
- Da prijenos podataka *Secret* objekata u unutarnjem prijenosu podataka (unutar sustava Kubernetes uvijek bude šifriran TLS-om,
- Da se Secret objekti uredno, redovito i automatski rotiraju.

3.7 Sigurnost mreže sustava Kubernetes

Prijelaz s fizičkih mreža koje koriste preklopnike, usmjerivače i mrežne kabele na virtualne mreže koje koriste softverski definirane mreže (eng. Software Defined Network; SDN) i virtualna sučelja ponekad uključuje određenu dozu nerazumijevanja. Naravno, principi ostaju isti, ali postoje različite specifikacije i najbolje prakse. Sustav Kubernetes ima vlastiti skup pravila koji se opisuju u nastavku.

Mrežni sloj sustava Kubernetes osmišljen je kako bi osigurao da različiti tipovi entiteta/objekata unutar sustava Kubernetes mogu međusobno komunicirati. Izgled Kubernetes infrastrukture ima, po svom dizajnu, mnogo međusobno povezanih dijelova.

Namespaces, spremnici (eng. Containers) i Pod-ovi različite su komponente sustava Kubernetes i postavljeni su na način da se međusobno razlikuju prema svojim zadaćama i funkcijama unutar sustava Kubernetes. Stoga, vrlo važan je strogo strukturiran plan njihove međusobne komunikacije.

Mrežni model sustava Kubernetes

Model mreže sustava Kubernetes specificira:

- Svaki Pod dobiva svoju IP adresu,
- Spremnici Pod modula dijele IP adresu modula i mogu slobodno međusobno komunicirati,
- Prema zadanim postavkama, Pod-ovi mogu komunicirati sa svim ostalim Pod-ovima u klasteru koristeći IP adrese Pod-ova i to bez korištenja NAT-a (eng. Network Address Translation),
- Izolacija resursa unutar Kubernetes klastera (ograničavanje s čime svaka grupa ili pojedini entitet može ili ne može komunicirati) definirana je pomoću mrežnih pravila (eng. Network Policy) koje se u nekim materijalima nazivaju i sigurnosna mrežna pravila (eng. Network Security Policies).

Kao rezultat toga, Pod-ove se može tretirati poput virtualnih strojeva (eng. Virtual Machines) ili hostova (svi oni imaju jedinstvene IP adrese), a spremnici unutar Pod-ova su poput procesa koji se izvode unutar VM-a ili hosta i pokreću se u istom mrežnom *NameSpaces* i dijele istu IP adresu.

Ovakav model uvelike olakšava migraciju aplikacija s VM-ova i hostova u Pod-ove kojima upravlja sustav Kubernetes.

Nadalje, budući da je izolacija definirana korištenjem mrežnih pravila, a ne strukturom mreže, mreža ostaje jednostavna za razumijevanje. Ovaj stil mreže ponekad se naziva i "ravna mreža" (eng. Flat Network).

Isto tako, moramo znati da, iako je vrlo rijetko potreban, sustav Kubernetes također podržava mogućnost mapiranja portova (eng. Port Maping ili Port Forwarding) hosta do Pod-ova ili pokretanja Pod-ova izravno unutar *NameSpaces* mreže domaćina dijeleći IP adresu hosta.

Implementacije Kubernetes mreže

Sustav Kubernetes ima ugrađenu mrežnu podrška, *kubenet. Kubenet* može pružiti osnovnu mrežnu povezanost. Međutim, češće je koristiti mrežne implementacije trećih strana koje se priključuju na Kubernetes pomoću CNI (Container Network Interface) API-ja.

Postoji puno različitih vrsta CNI dodataka, ali dva glavna su:

1. Mrežni dodaci (eng. Network Plugins), odgovorni su za povezivanje Pod-ova na mrežu sustava Kubernetes,
2. IPAM (eng. IP Address Management) dodaci, koji su odgovorni za dodjelu IP adresa određenim modulima unutar sustava Kubernetes.

Jedan od najpoznatijih i najboljih mrežnih dodataka je **Calico.** Calico je mrežno i mrežno sigurnosno rješenje otvorenog koda za spremnike, virtualne strojeve i sl.. Calico podržava širok raspon platformi uključujući sustav Kubernetes, OpenShift, Mirantis Kubernetes Engine (MKE), OpenStack i sl.

Calico model mrežne politike olakšava komunikaciju tako da jedini promet koji teče je onaj koji se definira od strane administratora sustava Kubernetes. Osim toga, s ugrađenom podrškom za Wireguard enkripciju postiže se sigurna mrežna komunikacija osiguravanjem prometa između Pod-ova.

Calico mehanizam za mrežnu politiku može primijeniti isti model politike na mrežnom sloju hosta i na uslužnom mrežnom sloju (eng. Service Mesh Layer), štiteći cjelokupnu infrastrukturu sustava Kubernetes od mogućih ugroza. Više o projektu Calico može se vidjeti na slijedećem linku: https://www.tigera.io/project-calico/

Kubernetes *Services*

Kubernetes *Services* pruža način apstrahiranja pristupa grupi Pod-ova kao mrežnu uslugu. Grupa Pod-ova obično se definira pomoću selektora oznaka (eng. Label Selector).

Unutar klastera, mrežna usluga tj. *Service* obično je predstavljena kao virtualna IP adresa, a *kube-proxy* uravnotežuje opterećenje veza s virtualnom IP-om preko grupe Pod-ova koji podržavaju uslugu.

Virtualni IP je vidljiv putem Kubernetes DNS-a. DNS ime i virtualna IP adresa ostaju nepromijenjeni tijekom životnog vijeka servisa tj. usluge (eng. Services), iako se Pod-ovi koji

podržavaju *Service* mogu stvoriti ili uništiti (nestati), a broj podova koji podržavaju uslugu može se mijenjati tijekom vremena.

Kubernetes *Services* također mogu definirati kako se usluzi pristupa izvan klastera, na primjer korištenjem:

- Priključak čvora (eng. Node Port), gdje se usluzi može pristupiti putem određenog priključka na svakom čvoru tj. samo unutar Kubernetes klastera,
- Balanser opterećenja (eng. Load Balancer), pruža virtualnu IP adresu preko koje se može pristupiti usluzi (eng. Services) izvan klastera.

Kada se koristi Calico u „on-prem“ implementacijama (dakle, ne u oblaku kao „KaaS“) također mogu se oglašavati IP adrese *Service-a,* omogućujući praktičan pristup uslugama bez prolaska kroz priključak čvora ili balanser opterećenja.

Mrežna politika u sustavu Kubernetes

Mrežna politika (eng. Network Policy) je primarni alat za osiguranje sigurnosti Kubernetes mreže. Omogućuje jednostavno ograničavanje mrežnog prometa u klasteru tako da je dopušten samo onaj promet koji se definira kao takav.

Povijesno gledano, mrežna sigurnost bila je osigurana dizajniranjem fizičke topologije mrežnih uređaja (preklopnici, usmjerivači, vatrozidi) i njihove konfiguracije. Drugim riječima, fizička topologija mreže definirala je sigurnosne granice mreže.

U prvoj fazi virtualizacije ista mreža i konstrukti mrežnih uređaja bili su virtualizirani, te su korištene iste tehnike za izradu specifičnih mrežnih topologija (virtualnih) mrežnih uređaja za osiguranje mrežne sigurnosti.

Dodavanje novih aplikacija ili usluga često se zahtijevalo za dodatnim dizajnom mreže kako bi se ažurirala topologija mreže i konfiguracija mrežnih uređaja kako bi se osigurala željena razina sigurnosti.

Nasuprot tome, Kubernetes mrežni model definira "ravnu" mrežu u kojoj svaki Pod može komunicirati sa svim ostalim Pod-ovima u klasteru koristeći IP adrese Pod-ova. Ovaj pristup uvelike pojednostavljuje dizajn mreže i omogućuje dinamičko raspoređivanje novih radnih opterećenja (eng. Workloads) bilo gdje u klasteru bez ovisnosti o dizajnu mreže.

U ovom modelu, umjesto da se mrežna sigurnost definira granicama mrežne topologije, ona se definira pomoću mrežnih pravila koja su neovisna o mrežnoj topologiji. Mrežna pravila su dodatno apstrahirana od mreže korištenjem *Label Selectors* kao primarnog mehanizma za definiranje koja radna opterećenja mogu komunicirati s kojim radnim opterećenjima, umjesto IP adresa ili raspona IP adresa.

Mrežna pravila sustava Kubernetes definirana su pomoću resursa sustava Kubernetes *NetworkPolicy.* Glavne značajke Kubernetes mrežnih pravila su:

- Pravila su ograničena unutar *Namspace* (tj. stvarate ih unutar konteksta određenog *Namespace* baš kao, npr. Pod-ovi)
- Pravila se primjenjuju na blokove pomoću *Label Selectors*
- Pravila mogu odrediti promet koji je dopušten do/od (eng. Ingress / Egress) drugih podova, prostora imena ili CIDR-ova
- Pravila mogu specificirati protokole (TCP, UDP, SCTP), imenovane portove ili brojeve portova

Sustav Kubernetes ne provodi mrežna pravila, već umjesto toga delegira njihovu provedbu mrežnim dodacima (eng. Network Plugins).

Većina mrežnih dodataka implementira glavne elemente Kubernetes mrežnih pravila, iako ne implementiraju svi svaku značajku specifikacije.

Prednosti korištenja i najbolje prakse Calico mrežnog dodatka za mrežnu politiku

1. **Puna podrška za Kubernetes mrežna pravila -** za razliku od nekih drugih implementacija mrežne politike, *Calico* implementira cijeli skup značajki Kubernetes mrežne politike.
2. **Raznovrsnija mrežna politika -** *Calico* mrežna pravila omogućuju još raznovrsniju kontrolu prometa od Kubernetes mrežnih pravila ukoliko je potrebna. Osim toga, Calico mrežna pravila omogućuju stvaranje pravila koja se primjenjuju na više imenskih prostora korištenjem resursa *GlobalNetworkPolicy.*
3. **Kombinacija Kubernetes i Calico mrežnih pravila -** mrežna pravila Kubernetes i *Calico* mogu se neprimjetno kombinirati. Jedan uobičajeni slučaj upotrebe za ovo je podjela odgovornosti između sigurnosnih i razvojnih timova.

   Na primjer, davanje dopuštenja putem RBAC timu za sigurnost klastera za definiranje Calico pravila i davanje dopuštenja putem RBAC timovima za razvojne usluge za definiranje mrežnih pravila u sustavu Kubernetes u njihovim definiranim *NameSpaces.*

4. **Sposobnost zaštite hostova i VM-ova -** budući da se pravila *Calico* mogu provoditi na sučeljima hosta, mogu se koristiti i za zaštitu Kubernetes čvorova (ne samo Pod-ova), uključujući, npr., ograničavanje pristupa portovima čvorova izvan klastera.
5. **Ingress and Egress -** preporuka jest da svaki modul bude zaštićen ulaznim pravilima mrežne politike koja ograničavaju što je dopušteno povezati s modulom i na kojim priključcima (eng. Ports).

   Najbolja praksa također govori da je potrebno definirati izlazna pravila mrežne politike koja ograničavaju odlazne veze koje dopuštaju sami moduli. Ulazna pravila štite grupu od napada izvan grupe. Pravila izlaza pomažu u zaštiti svega izvan modula ako modul bude kompromitiran, smanjujući površinu napada kako bi se kretao bočno (istok-zapad) ili kako bi spriječio napadača da filtrira ugrožene podatke iz klastera (sjever-jug).
6. **Sheme politika -** zbog fleksibilnosti mrežne politike i označavanja, često postoji više različitih načina označavanja i pisanja politika kojima se može postići isti cilj. Jedan od najčešćih pristupa je imati mali broj globalnih pravila koja se primjenjuju na sve grupe, a zatim jednu specifičnu politiku grupe koja definira sva ulazna i izlazna pravila koja su specifična za tu grupu.
7. **Zadano odbijanje (eng. Default Deny) -** jedan pristup osiguravanju poštivanja najboljih praksi je definiranje zadanih mrežnih pravila zabrane. Oni osiguravaju da će promet biti odbijen ako nije definirano nijedno drugo pravilo koje izričito dopušta promet prema/iz Pod-a. Kao rezultat toga, svaki put kada razvojni tim postavi novu jedinicu, prisiljeni su također definirati mrežna pravila za grupu. Može biti korisno koristiti Calico *GlobalNetworkPolicy* za ovaj slučaj (umjesto potrebe za definiranjem pravila svaki put kada se kreira novi *Namespace*) i uključiti neke iznimke od zadanog odbijanja (na primjer kako bi se modulima omogućilo pristup DNS servisu). Na primjer, može se upotrijebiti sljedeće pravilo za zadano odbijanje cjelokupnog (ne-sistemskog) Pod prometa osim DNS upita za kube-dns/core-dns.

Primjer mrežne politike (*NetworkPolicy)*:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: front-end-aplikacija
  namespace: app_namespace
spec:
  podSelector:
    matchLabels:
      app: back-end-aplikacija
  ingress:
    - from:
      - podSelector:
          matchLabels:
            app: front-end-aplikacija
    ports:
    - protocol: TCP
      port: 443
  egress:
    - to:
      - podSelector:
          matchLabels:
            app: database
    ports:
    - protocol: TCP
      port: 27017
```

Primjer globalne mrežne politike (*GlobalNetworkPolicy*):

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: globalna-app-politika
spec:
  namespaceSelector: has(projectcalico.org/name) && projectcalico.org/name not in {"kube-system", "calico-system", "calico-apiserver"}
  types:
  - Ingress
  - Egress
  egress:
    - action: Allow
      protocol: UDP
      destination:
        selector: k8s-app == "kube-dns"
        ports:
        - 53
```

U slijedećem poglavlju, **4. Nadzor sustava Kubernetes** opisati će se načini nadzora sustava Kubernetes, ključne metrike u praćenju sustava Kubernetes tj. Kubernetes klastera te alate za nadzor Kubernetes klastera.

4 NADZOR SUSTAVA KUBERNETES

Nadzor sustava Kubernetes (eng. Monitoring) je oblik izvješćivanja koji pomaže u proaktivnom upravljanju Kubernetes klasterima na nekoliko razina. Nadgledanje Kubernetes klastera ponajprije olakšava praćenje i upravljanje kontejnerskom infrastrukturom praćenjem korištenja resursa klastera uključujući memoriju, CPU, pohranu, mrežu i sl., ali služi i proaktivnom praćenju i nadzorom sigurnosti sustava Kubernetes.

Sistemski administratori Kubernetes klastera mogu pratiti i primati upozorenja od strane Kubernetes klastera u situacijama npr. ako željeni broj Pod-ova ne radi, ili ako se iskorištenost resursa približava kritičnim granicama ili kada kvarovi ili pogrešna konfiguracija uzrokuju da Pod-ovi ili čvorovi ne mogu sudjelovati u klasteru. Takove situacije mogu ukazivati i na sigurnosne probleme i/ili probleme s konfiguracijom samog Kubernetes klastera, njegovih dijelova, modula, objekata i sl.

4.1 Ključne metrike u praćenju sustava Kubernetes

Kubernetes Metrics Server agregira podatke prikupljene iz *kubelet* na svakom čvoru, propuštajući ih kroz *Metrics API*, koji se zatim može kombinirati s brojnim alatima za vizualizaciju. Ključne metrike koje treba uzeti u obzir za praćenje uključuju:

- **Podaci o stanju klastera** - uključujući i stanje i dostupnost Pod-ova
- **Status svih klaster čvorova** - stanje memorije, diska, procesora, dostupnost mreže i sl.
- **Dostupnost Pod-ova** - nedostupni Pod-ovi mogu ukazivati na sigurnosne probleme ili probleme s konfiguracijom
- **Korištenje memorije** - na razini Pod-ova i čvorova unutar Kubernetes klastera
- **Iskorištenje diska** - nadzor mogućeg nedostataka prostora za datotečni sustav
- **Iskorištenje CPU-a** - nadzor količine CPU resursa dodijeljenog pojedinom Pod-u.

U današnje vrijeme u kojem se događa eksponencijalan rast broja kontejnera u sustavima kao što je sustav za orkestraciju Kubernetes donio je mnoge prednosti programerima, *DevOps, DevSecOps* i IT timovima općenito.

Međutim, fleksibilnost i skalabilnost koju sustav Kubernetes donosi u implementaciji kontejnerskih aplikacija također predstavlja nove izazove. Kako one sigurnosne prirode tako i one koje se odnose na nadzor i administraciju sustava kao što je sustav za orkestraciju Kubernetes.

Budući da više ne postoji korelacija „1-na-1“ između aplikacije i poslužitelja na kojem radi, praćenje stanja aplikacija apstrahiranih u kontejnerima i ponovno apstrahiranih unutar sustava Kubernetes može predstavljati veliki problem i izazov bez upotrebe odgovarajućih alata.

Nadzor Kubernetes klastera

Prilikom nadzora Kubernetes klastera dobiva se potpuni pregled svih područja, kao što su: stanje svih Pod-ova, stanje čvorova unutar klastera i stanje postavljenih aplikacija.

Ključna područja za praćenje na razini Kubernetes klastera uključuju:

- **Opterećenje čvora (eng. Node Load):** Praćenje opterećenja na svakom čvoru sastavni je dio praćenja učinkovitosti klastera s time da se neki se čvorovi koriste više od drugih. Ponovno balansiranje raspodjele opterećenja ključno je za održavanje učinkovitih radnih opterećenja. To se može učiniti putem objekta *DaemonSets.*
- **Neučinkoviti Pod-ovi (eng. Unsuccessful pods):** Pod-ovi mogu biti u stanju *Fail* i *Abort*. To je normalan dio procesa unutar Kubernetes klastera. Kada je Pod koji bi trebao raditi u stanju *Fail* ili je neaktivan, važno je istražiti razlog/e iza takovih anomalija.
- **Korištenje/opterećenje klastera (eng. Cluster usage):** Nadgledanje infrastrukture klastera omogućuje prilagodbu broja čvorova u upotrebi i raspodjelu resursa za učinkoviti rad svih radnih procesa (eng. Workloads). Vidljivost resursa koji se distribuiraju omogućuje povećanje ili smanjenje i izbjegavanje troškova dodatne infrastrukture. U skladu s tim, važno je postaviti ograničenje upotrebe memorije i CPU-a s obzirom na potrebne radnih procesa unutar Kubernetes klastera.

Praćenje Kubernetes Pod-ova

Praćenje klastera pruža globalni pogled na okruženje sustava Kubernetes tj. Kubernetes klastera. Prikupljanje podataka iz pojedinačnih Pod-ova također je vrlo važno. Ono otkriva stanje pojedinačnih Pod-ova i radnih procesa u kojima se nalaze, pružajući jasniju sliku performansi Pod-ova na granularnoj razini.

Ključna područja za praćenje na razini Pod-ova uključuju:

- **Ukupne instance svih Pod-ova (eng. Total Pod Instances):** u svakom trenutku mora postojati dovoljan broj instanci Pod-ova kako bi se osigurala visoka dostupnost. Na ovaj način propusnost hostinga nije uzalud potrošena, no potrebno je razmotriti da se ne pokrene previše dodatnih instanci Pod-ova.
- **Stvarne instance svih Pod-ova (eng. Actual Pod Instances):** Posebno obratiti pažnju na praćenje broja instanci za svaki Pod koji je pokrenut u odnosu na ono što se očekuje da će biti pokrenuto. Na taj način može se otkriti kako preraspodijeliti resurse s ciljem da se postigne željeno stanje u smislu broja Pod instanci. *ReplicaSet* mogu biti pogrešno konfigurirani s različitim metrikama, stoga je važno redovito ih analizirati i ažurirati.
- **Implementacije Pod-ova (eng. Pod Deployments):** Praćenje razvoja Pod-ova omogućuje pregled svih pogrešnih konfiguracija koje bi mogle smanjiti dostupnost Pod-ova. Od ključne je važnosti pratiti kako se resursi distribuiraju po radnim čvorovima u Kubernetes klasteru.

4.2 Alati za nadzor Kubernetes klastera

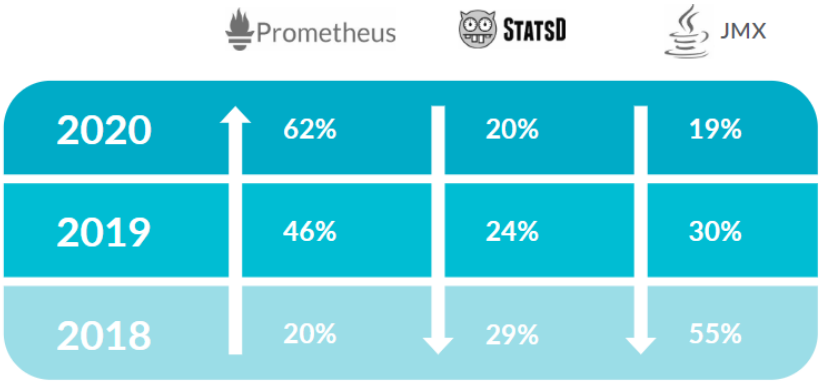
Kao što je već u nekoliko navrata rečeno, sustav Kubernetes je vrlo složeno okruženje. Aplikacije u kontejnerima mogu se distribuirati u više okruženja (i to na više fizičkih lokacija) što predstavlja još veći izazov glede nadzora sigurnosti i općenito rada i funkcioniranja sustava Kubernetes.

Rješenja za praćenje moraju biti u stanju agregirati metrike iz cijelog distribuiranog okruženja, ali i nositi se s npr. karakteristikama relativno kratkotrajnom prirodom kontejnerskih resursa (npr. Pod-ova).

Sustavi *Prometheus* i *Grafana*

Prometheus je jedan od najpopularnijih alata za nadzor i praćenje sustava Kubernetes. Razvio ga je *SoundCloud* prije nego što je doniran *Cloud Native Computing Foundation* (CNCF).

Prema istraživanju tvrtke SysDig sustav *Premetheus* konstantno raste s obzirom na broj implementacija što govori i grafikon na **Slici 21.**



Slika 21. Postotak implementacija sustava Prometheus prema istraživanju tvrtke SysDig (Izvor: https://dig.sysdig.com/c/pf-2021-container-security-and-usage-report?x=hJvo1P#page=1)

Prometheus pruža mogućnosti postavljanja upozorenja kao i detaljne metrike s analizama za sustav Kubernetes i Docker. Dizajniran je za nadzor mikroservisa temeljenih na spremnicima i aplikacija koje se izvode u sustavima Kubernetes u velikom broju. Prometheus se često koristi u kombinaciji s sustavom Grafana koji se koristi za vizualizaciju dobivenih metrika tj. podataka.

Slika 22. Logo sustava Prometheus i Grafana (Izvori: https://prometheus.io/; https://grafana.com/)

Slika 23. Primjer nadzorne ploče u sustavu Grafana (Izvor: https://grafana.com)

Grafana je platforma otvorenog koda i služi za vizualizaciju metrika i analitika te pruža nekoliko ugrađenih nadzornih ploča (eng. Dashboard) i to za: Kubernetes klaster, Kubernetes Node, Pod/Container i Deployment.

Slika 24. Primjer nadzorne ploče u sustavu Grafana (Izvor: https://grafana.com)

Administratori sustava Kubernetes mogu kreirati nove, prilagođene nadzorne ploče u sustavu Grafana koristeći informacije dobivene iz sustava Prometheus te na taj način sustav Grafana prilagoditi specifičnostima vlastitog sustava tj. okruženja. To sustavu Grafana u kombinaciji sa sustavom Prometheus daje veliku snagu.

Sustav *Lens IDE* / *Open Lens*

Lens je prvenstveno IDE, integrirano razvojno okruženje (eng. Integrated Development Environment) koje korisnicima omogućuje povezivanje i upravljanje više Kubernetes klastera na Mac, Windows i Linux platformama. Izvorno ga je razvila „Kontena“, finska startup tvrtka, a kasnije ga je kupio Mirantis i otvorio ga pod licencom MIT-a.

Lens pruža jako dobar uvid u stanje sustava Kubernetes, prvenstveno Kubernetes klastera te ostalih komponenti sustava.

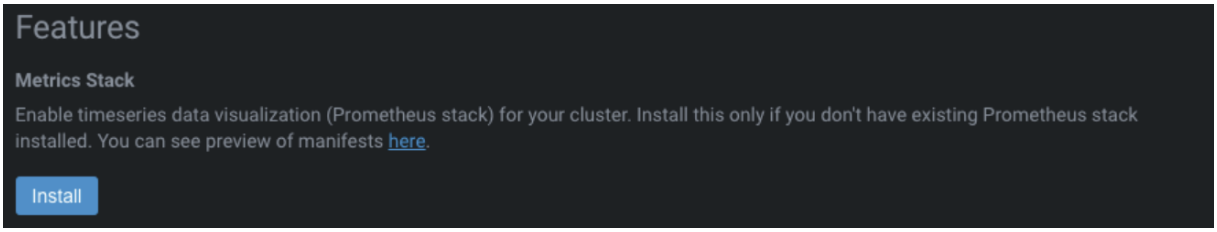
Lens pruža grafičko sučelje koje:

- Korisnicima omogućuje implementaciju i upravljanje Kubernetes klasterima izravno s konzole (omogućuje istodobno praćenje više Kubernetes klastera što mu je jedna od vrlo važnih karakteristika),
- Omogućuje ugrađene nadzorne ploče s ključnim metrikama i uvidima u resurse koji se izvode u Kubernetes klasteru, uključujući konfiguracije, umrežavanje, pohranu i kontrolu pristupa.

Lens omogućuje praćenje i upravljanje jednim ili više Kubernetes klastera jednostavno i brzo pomoću aplikacije otvorenog koda za stolna (eng. Desktop) računala. Instalacija je vrlo jednostavna i podržava sve važnije operativne sustave kao što su Windows, Mac ili Linux.

Lens objavljuje nova ažuriranja svaki mjesec. Na taj način korisnici Lens sustava dobivaju najnovije značajke na mjesečnoj bazi.

Lens koristi *Prometheus stack* za dobivanje svih metrika specifičnih za klaster. To bi značilo da korisnici Lens sustava moraju imati postavljen i konfiguriran Prometheus. Ova značajka može se postaviti i direktno kroz instalaciju sustava Lens, ukoliko ne postoji postavljen i konfiguriran sustav Prometheus.



Slika 25. Instalacija sustava Prometheus unutar instalacije sustava Lens (Izvor: https://k8slens.dev/)

Ukoliko je Prometheus sustav instaliran, potrebno je izvršiti neka manja podešavanja kako bi osigurali da Lens u potpunosti točno prikazuje podatke iz Kubernetes klastera.

Također, potrebno je instalirati „*Kubernetes Metrics Server*“ jer Lens isti koristi za prikaz nekih podataka na razini čvorova i Pod-ova.

Koje probleme rješava sustav Lens?

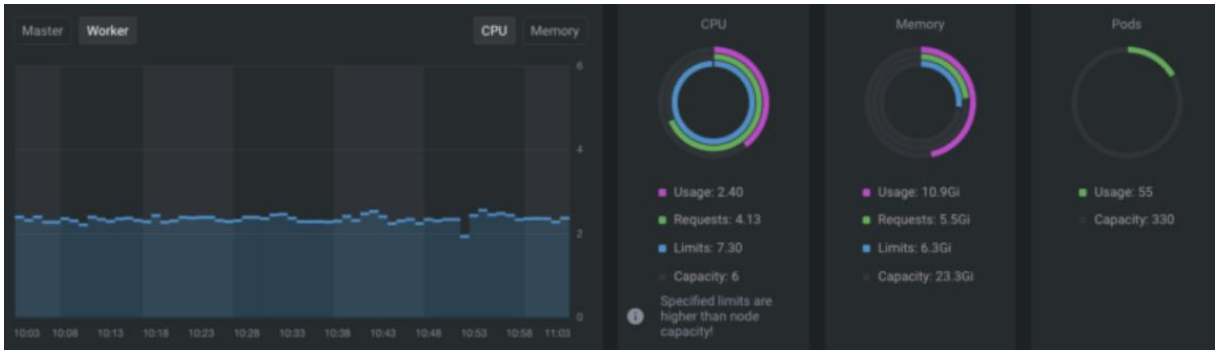
DevOps timovi izrađuju aplikacije pomoću složenih distribuiranih arhitektura. U ozbiljnijim okruženjima, aplikacije se u pravilu razvijaju lokalno i postavljaju u niz Kubernetes klastera za npr. razvoj, testiranje, i proizvodnju koristeći kontrolu verzija (eng. Version Control) i CI/CD alate (eng. Continuous Integration; CI), Continuous Delivery; CD).

Administratori Kubernetes klastera koriste iste alate za izgradnju i skaliranje klastera, međutim, postoji nedostatak koordinacije u upravljanju većeg broja klastera.

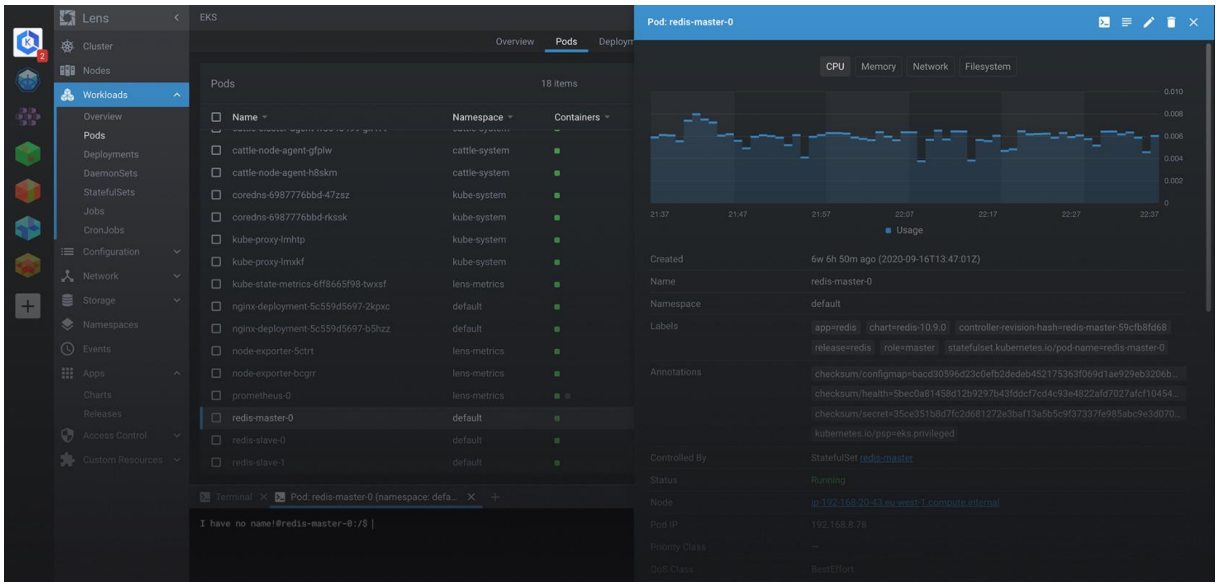
Moderna poduzeća daju prednost malim klasterima specifičnim za određene zadatke u odnosu na velike, produkcijske Kubernetes klastere koji objedinjuju više različitih zadataka, okolina i sl. na jednom mjestu.

Rezultat toga jest da članovi tima moraju rukovati većim brojem klastera koji u velikim produkcijskim okolinama prelazi broj od nekoliko stotina. Problem je u tome što CLI-ovi koji komuniciraju s klasterima i koriste mnogo različitih konfiguracijskih datoteka, što otežava upravljanje složenim, raznolikim nizom metoda i konteksta.

Upravljanje infrastrukturom iz naredbenog retka (eng. Command Line Interface; CLI) sporo je i podložno pogreškama, osobito pri skaliranju aplikacija i klastera. Isto tako, konfiguracije se razlikuju, postaju sve veće i kompleksnije i postaje ih sve teže pratiti.



Slika 26. Primjer nadzorne ploče sustava Lens (Izvor: https://k8slens.dev/)



Slika 27. Primjer nadzorne ploče sustava Lens (Izvor: https://k8slens.dev/)

Lens objedinjuje alate i informacije potrebne za rad s različitim kontekstima i zadacima i pomaže u rješavanju problema tako što:

- Omogućuje automatsko dodavanje Kubernetes klastera,
- Automatsko otkrivanje lokalnih *kubeconfig* datoteka i omogućuje upravljanje klasterima na bilo kojoj infrastrukturi,
- Omogućuje organiziranje velikog broja Kubernetes klastera, za rješavanje problema širenja klastera (u vertikalnom ili horizontalnom nivou),
- Upravljanje više *kubectl* verzija - Lens instalira potrebnu verziju za svaki klaster.
- Automatsko ograničavanje interakcija s ograničenjima korištenjem RBAC-a, tako da korisnici pristupaju samo dopuštenim resursima unutar Kubernetes klastera,
- Automatsko instaliranje Prometheus instanci u bilo kojem *NameSpace-u* radi pružanja sve potrebne metrike na tom nivou.

Sigurnosna značajka sustava Lens

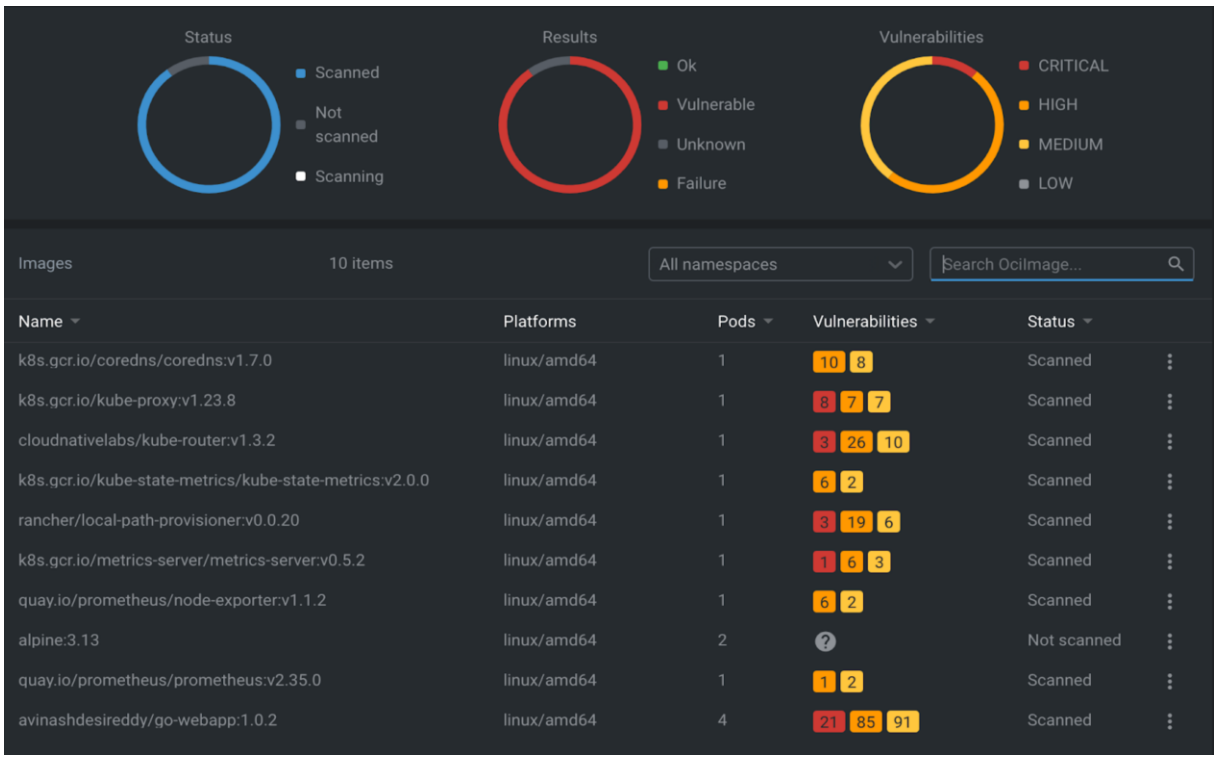
Spremnici (eng. Containers) mogu učiniti razvoj, pakiranje (eng. Packaging) i postavljanje aplikacija jeftinijima i lakšima za upravljanje ukoliko se koristimo sa sustavom za orkestraciju za velikim brojem kontejnera što je slučaj sa svim većim Kubernetes sustavima.

Budući da su spremnici standardizirani, kod aplikacije, ovisnosti, biblioteke i vrijeme izvođenja (eng. Runntime) upakirani su u spremnike i slike (eng. Images) kako bi se povećala agilnost i pomoglo u brzom pružanju usluge krajnjim korisnicima.

No, dok spremnici programerima pružaju određene prednosti, oni također povećavaju rizik za poslovanje jer programer ima pristup širokom rasponu javnih alata i biblioteka za implementaciju određenih značajki za aplikaciju.

Za programere je ključno da imaju uvid u ranjivosti rano u procesu ciklusa razvoja softvera/aplikacija (eng. Software Development Life Cycle; SDLC) kako bi smanjili površinu napada i učinili aplikaciju što sigurnijom.

Značajka *Lens Security,* dolazi s ugrađenim mogućnostima skeniranja slike spremnika (eng. Container Image) i omogućava trenutno identificiranje ranjivosti prisutnih u toj slici spremnika.



Slika 28. Primjer nadzorne ploče „Lens Security“ sustava Lens (Izvor: https://k8slens.dev/)

Kako značajka *Lens Security* funkcionira i kako programeri i DevOps timovi mogu imati koristi od nje, potrebno je navesti uobičajene korake uključene u prepoznavanje i popravljanje ranjivosti prilikom razvoja aplikacijskih rješenja u poslovnoj organizaciji.

Uobičajena praksa koja se primjenjuje u najvećem broju poslovnih organizacija, a odnosi se na razvoj programskih/aplikativnih rješenja protječe otprilike na slijedeći način:

1. Programer piše kod za neku aplikaciju,
2. Programer postavlja svoj kod u neki repozitorij koda (GitLab, GitHub ili lokalno),
3. Pokreće se CI/CD proces, izrađuje i postavlja slika (eng. Image) u registar i postavlja se u Kubernetes klaster,
4. Programeri (i drugi timovi uključeni u razvoj) potvrđuju implementaciju u npr. okruženju „Razvoj“,
5. Aplikacija se promiče u viša okruženja (npr. Test, Prod),
6. Sigurnosni ili operativni timovi procjenjuju rizik u višim okruženjima (obično u okruženju “Test“ pa zatim i u okruženju „Prod“),
7. Budući da nije uobičajeno skenirati aplikacije ili spremnike u nižim okruženjima, sigurnosni tim stvara izvješće s popisom ranjivosti i razvojnim timovima koji su odgovorni za njihovo rješavanje,
8. Problem se ispituje i rješava u sljedećem izdanju aplikacije.

Ovaj proces je opsežan, dugotrajan i nije pogodan za razvojni tim jer se ranjivosti identificiraju tek u posljednjim fazama SDLC-a.

Sustav Lens uz dodatak *Lens Security*, programerima i DevOps timovima omogućuje skeniranje slika spremnika i pregled izvješća o ranjivostima vrlo rano u procesu razvoja.

Na taj način ranjivosti se detektiraju u začetcima razvoja programskih/aplikativnih rješenja što razvojnom timu štedi vrijeme i smanjuje moguću površinu napada, a programska/aplikativna rješenja čini sigurnijima.

Drugim riječima, značajka sustava Lens, *Lens Security,* izravno je ugrađena u radni tijek razvoja (eng. Development Workflow) programskih tj. aplikativnih rješenja, pomaže identificirati sigurnosne ranjivosti rano u procesu razvoja istih i ima minimalne ili nikakve smetnje od strane sigurnosnog ili operativnog tima.

U slijedećem poglavlju, **5. Zapisivanje u sustavu Kubernetes** opisati će se na koji način funkcionira zapisivanje (eng. Logging) u sustavu Kubernetes te navesti neke od alata za zapisivanje sustava Kubernetes tj. alate za prikupljanje, pretraživanje i analizu zapisa (eng. Logs).

5 ZAPISIVANJE U SUSTAVU KUBERNETES

Zapisivanje (eng. Logging) u sustavu Kubernetes uvelike se razlikuje od zapisivanja na tradicionalnim poslužiteljima ili virtualnim strojevima. To se očituje uglavnom zbog načina na koji sustav Kubernetes upravlja svojim aplikacijama tj. Pod-ovima.

Kada se aplikacija „sruši“ na virtualnom računalu, zapisi su i dalje dostupni dok ih se ne izbriše. U sustavu Kubernetes, kada su Pod-ovi izbačeni, srušeni, izbrisani ili pak prestanu raditi na drugom čvoru, zapisnici iz spremnika nestaju. Možemo reći da sustav Kubernetes na taj način „čisti za sobom“.

U tom slučaju gube se sve informacije o tome zašto je došlo do zastoja i/ili anomalije. Takova, prolazna priroda zadanog zapisivanja u sustavu Kubernetes čini se ključnom za implementaciju centraliziranog rješenja za upravljanje logovima.

Nadalje, ne smijemo zanemariti i visoko distribuiranu i dinamičnu prirodu sustava Kubernetes. U produkcijskom okruženju sustav Kubernetes biti će (vjerojatno) sačinjen od većeg broja strojeva koji će sačinjavati Kubernetes klaster od kojih će svaki od njih imati više spremnika koji se mogu srušiti u bilo kojem trenutku.

S druge strane, Kubernetes klasteri po svojoj prirodi još više pridonose složenosti uvođenjem novih slojeva koje je potrebno nadzirati, a svaki od njih generira vlastitu vrstu zapisa tj. logova.

5.1 Kako funkcionira zapisivanje u sustavu Kubernetes

Postoje različiti načini kojima se mogu prikupljati zapisi u sustavu Kubernetes:

1. **Osnovno bilježenje koristeći „Stdout“** (eng. Standard Output) **i „Stderr“** (eng. Standard Error Output)

U tradicionalnim poslužiteljskim okruženjima, zapisnici pojedinih aplikacija zapisuju se u datoteke kao što je npr. */var/log/app.log*. Ovi se zapisnici mogu pregledavati „ručno“ na svakom poslužitelju ili ih može prikupiti agent koji ih onda prosljeđuje na središnje mjesto za analizu logova i pohranu istih.

Međutim, u sustavu Kubernetes, zapisi se moraju prikupiti za više Pod-ova (aplikacija), preko više čvorova u klasteru, što ovu metodu prikupljanja zapisa ne čini baš optimalnom.

Upravljanje sa više datoteka dnevnika (zapisa) za više spremnika na više poslužitelja u klasteru zahtijeva novi i mnogo zahtjevniji zadatak koji iziskuje vrlo specifične radnje i procese.

Umjesto toga, zadani okvir sustava Kubernetes za bilježenje preporučuje snimanje standardnog izlaza, *stdout* i standardnog izlaza pogreške, *stderr* iz svakog spremnika na čvoru u datoteku dnevnika. Ovom datotekom upravlja sam sustav Kubernetes i obično je ograničena na zadanih 10 MB zapisa. Zapisnici određenog spremnika mogu se vidjeti pokretanjem naredbe:

 # kubectl logs <naziv spremnika>.

2. **Korištenje konfiguracije zapisivanja na razini aplikacije** (eng. Application Level Logging Configuration)

Za razliku od konfiguracije logiranja na razini Kubernetes klastera, može se koristiti konfiguracija zapisivanja na razini same aplikacije.

To podrazumijeva da svaki spremnik ima vlastitu konfiguraciju zapisivanja, što samo po sebi čini zapisivanje/logiranje vrlo kompleksnim, teškim i podložno pogreškama u konfiguraciji.

Nadalje, bilo kakve promjene u konfiguraciji na razini spremnika zahtijevale bi ponovno postavljanje spremnika koji se žele nadzirati što dodatno stvara probleme.

3. **Korištenje agenata za bilježenje**

Agenti za bilježenje alati su koji prikupljaju zapisnike iz sustava Kubernetes i šalju ih na središnje mjesto gdje se isti onda pohranjuju i obrađuju.

Agenti su u pravilu vrlo „lagani“ spremnici koji imaju pristup direktoriju sa zapisima iz svih spremnika aplikacija na čvoru.

Ovo način je najjednostavniji i predstavlja najbolji rješenje budući da ne utječe na postavljene aplikacije i u potpunosti je skalabilan, bez obzira koliko čvorova sadrži Kubernetes klaster.

Također, implementacija ovog načina prikuplja zapisa je relativno jednostavna za postavljanje i u pravilu, uz pretpostavku ispravne konfiguracije agenata, ne utječe na rad i funkcioniranje samog sustava Kubernetes, tj. Kubernetes klastera.

Arhitektura Kubernetes zapisivanja i vrste Kubernetes zapisa

Kao što je ranije spomenuto, postoji vrlo mnogo slojeva zapisivanja u sustavu Kubernetes, a svi sadrže različite, ali jednako korisne informacije ovisno o „scenariju“ tj. arhitekturi samog sustava Kubernetes, aplikacija koje se na njemu izvršavaju i mnogih drugih detalja.

Unutar sustava Kubernetes možemo identificirati tri vrste zapisa:

1. **Zapisnici Pod-ova (kontejnera),**
2. **Zapisnici Kubernetes čvorova,**
3. **Zapisnici Kubernetes klastera.**

1. **Zapisnici Kubernetes Pod-ova** (kontejnera)

Zapisnike Kubernetes Pod-ova nalaze se u sljedećim direktorijima svakog radnog čvora.

/var/log/containers - svi dnevnici spremnika prisutni su na jednom mjestu.

/var/log/pods/ - na ovoj lokaciji nalaze se dnevnici spremnika su organizirani u zasebnim Pod mapama.

/var/log/pods/<namespace>_<pod_name>_<pod_id>/<container_name>/.

Svaka mapa pojedinog Pod-a sadrži mapu pojedinačnog spremnika i odgovarajuću datoteku dnevnika. Svaka mapa ima shemu imenovanja (eng. Naming Scheme)

2. **Zapisivanje Kubernetes čvorova** (eng. Nodes)

Sve što aplikacija razvijena na bazi spremnika zapiše u *stdout* ili *stderr* prenosi se negdje putem spremnika, a u slučaju korištenja Docker-a kao kontejner runtime-a, na primjer, u upravljački program za bilježenje. Ti se zapisnici obično nalaze u direktoriju */var/log/containers* na svakom hostu tj. čvoru gdje se kontejner izvršava.

Ako se spremnik ponovno pokrene, *kubelet* vodi zapisnike na čvoru. Kako bi se spriječilo da zapisnici popune sav raspoloživi prostor na čvoru (hostu), sustav Kubernetes ima postavljenu politiku rotacije dnevnika. Stoga, kada je Pod izbačen iz čvora, svi odgovarajući spremnici također su izbačeni, zajedno sa svojim zapisima.

Ovisno o operativnom sustavu i uslugama, postoje različiti zapisnici na razini čvora koji se mogu prikupiti, kao što su npr. zapisnici *kernela* ili *systemd* zapisnici.

3. Zapisivanje Kubernetes klastera

Dnevnici klastera sustava Kubernetes odnose se na sam sustav Kubernetes i zapise svih njegovih komponenti. Razlikujemo komponente koje se pokreću u spremniku i komponente koje se ne pokreću u spremniku. Svaki ima svoju ulogu, dajući uvid u stanje sustava Kubernetes tj. Kubernetes klastera.

Na primjer, *kube-scheduler, kube-apiserver* i *kube-proxy* rade unutar spremnika, dok *kubelet* i *runtime* spremnika (eng. Container Runtime) rade na razini operativnog sustava, obično kao *systemd* servis kada govorimo o sustavu Linux što je situacija u većini slučajeva.

Prema zadanim postavkama, komponente sustava izvan spremnika pišu datoteke u dnevnik, dok komponente koje se izvode u spremnicima pišu u direktorij /var/log.

Međutim, postoji opcija za konfiguriranje spremnika za spremanje zapisa na određenu (željenu) tj. proizvoljnu lokaciju.

Sustav Kubernetes ne pruža izvorno rješenje (eng. Native Solution) za bilježenje na razini klastera. Međutim, dostupni su i drugačiji pristupi ovom problemu kao što su:

- Korištenje agenta za bilježenje na razini čvora koji radi na svakom čvoru,
- Dodavanje *sidecar container* tj. spremnika za bilježenje unutar modula aplikacije,
- Ili izlaganje zapisa izravno iz aplikacije.

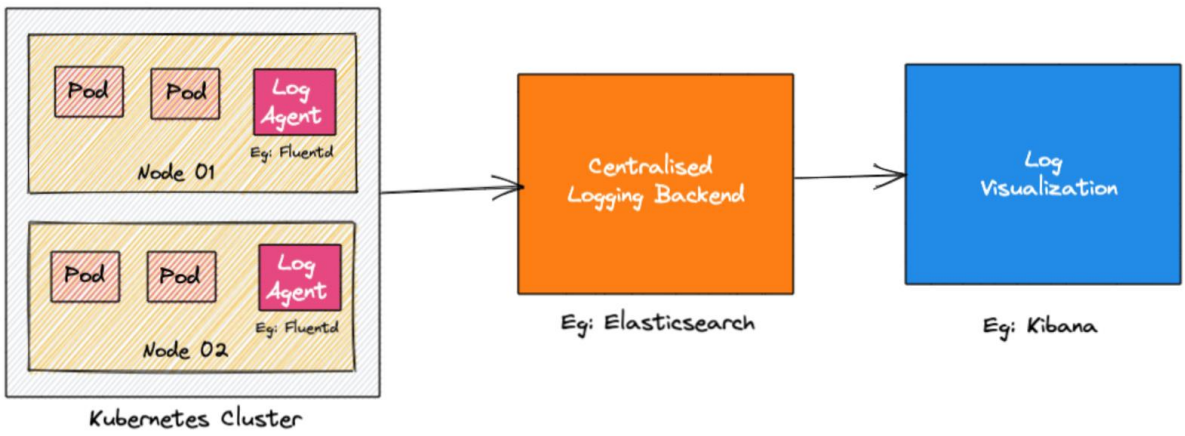
5.2 Alati za zapisivanje sustava Kubernetes - prikupljanje i analiza zapisa

Ako uzmemo Kubernetes klaster kao cjelinu, zapisivanju (eng. Logging) kao procesu moramo pristupiti na način da pokušamo centralizirati sve zapise. Kao što je i rečeno ne postoji zadana funkcija sustava Kubernetes za centralizaciju zapisa.

Drugim riječima, za učinkoviti pristup rješavanju problema prikupljanja i centraliziranja procesa zapisivanja u sustavu Kubernetes potrebno je postaviti centralizirani način (sustav) za bilježenje kao što je npr. sustav **Elasticsearch.**

Slika 29. Logo sustava ElasticSearch i Kibana (Izvor: https://www.elastic.co/)

Slika 30. prikazuje arhitekturu sustava Kubernetes sa postavljenim agentom i sustavom za zapisivanja na visokoj razini koristeći sustave Fluentd, ElasticSearch i Kibana.



Slika 30. Arhitekturu sustava Kubernetes u korelaciji sa sustavom ELK
(Izvor: https://devopscube.com/kubernetes-logging-tutorial/)

Osnovne komponente ovog načina sustava zapisivanja su:

1. **Agent za zapisivanje (eng. Logging Agent):** Agent za bilježenje koji se pokreće kao *Demonset* i kao takav, pokreće se na svim Kubernetes čvorovima. Agent kontinuirano šalje zapisnike u pozadinski sustav za zapisivanje. agent može biti npr. *Fluentd*, *Logstash* ili *Filebeat* (ovisno o konfiguraciji ELK),
2. **Pozadinski sustav za zapisivanje (eng. Logging Backend):** Centralizirani sustav koji služi za pohranjivanje, pretraživanje i analizu podataka zapisa. Klasičan primjer je *Elasticsearch,*
3. **Sustav za vizualizaciju zapisa (eng. Log Visualization):** Alat za vizualizaciju podataka zapisa u obliku nadzornih ploča (eng. Dashboards). Na primjer, sustav *Kibana.*

ELK stack (eng. **E**lasticsearch-**L**ogstash-**K**ibana)

Glavna svrha ELK sustava je objediniti zapise i pripremiti ih za pretraživanje i analizu. Arhitektura mikroservisa zahtijeva način prikupljanja i pretraživanja zapisa (logova) u svrhu otklanjanja pogrešaka, uvida u sustav te analizu (povijesno gledano) nakon nekog npr. sigurnosnog incidenta. Stoga, možemo reći da ELK pomaže u skupljanju zapisa i istraživanju kroz te zapise.

ELK Stack (eng. **E**lasticSearch-**L**ogstash-**K**ibana) sastoji se od četiri glavne komponente:

1. ElasticSearch,
2. Filebeat,
3. Logstash,
4. Kibana.

Elasticsearch je baza podataka koja pohranjuje sve zapise.

Filebeat je vrlo važna komponenta i radi kao alat za „izvoz zapisa“ (eng. Log Exporter). Izvozi i prosljeđuje zapise u Logstash.

Logstash je alat za unos podataka. Unosi podatke (zapisnike) iz različitih izvora i obrađuje ih prije slanja u Elasticsearch bazu podataka.

Kibana je platforma za vizualizaciju i koristi se za postavljanje upita nad Elasticsearch bazom podataka.

U slijedećem poglavlju, **6. Revizija sustava Kubernetes** opisati će se način na koji možemo dodatno zaštiti sustav Kubernetes odnosno Kubernetes klaster primjenom politike revizije (eng. Audit Policy), na koji način konfigurirati sustav pravila revizije te kako iskoristiti, analizirati i upotrijebiti na taj način dobivene podatke.

6 REVIZIJA SUSTAVA KUBERNETES

Kako popularnost sustava za orkestraciju Kubernetes raste svakim danom, izazov održavanja sigurnosnih postavki višestruko se povećao. Sa svakim novim spremnikom koji se dodaje u sustav, širi se područje napada, a tako i broj ulaznih točaka za zlonamjerne aktivnosti od strane raznih „vanjskih“ ili „unutarnjih“ aktera.

Dakle, ako se sustav skalira bez provjere sigurnosti, kritične podatci se izlažu neovlaštenim korisnicima i time se sustav i organizaciju izlaže ozbiljnom sigurnosnom riziku.

Iz tih razloga koriste se i revizijski zapisnici (eng. Audit Logs) sustava Kubernetes. Revizijski zapisnici omogućuju smanjivanje rizika od neželjenog pristupa i pogrešnih konfiguracija, uz zaštitu organizacije od zlouporabe resursa sustava Kubernetes.

Revizijski zapisnici kronološki su zapisi radnji izvršenih putem Kubernetes API-a. Revizijski zapisnici korisni su i za analizu onoga što se izvršava unutar Kubernetes klastera u stvarnom vremenu što predstavlja veliku prednost.

Revizijski zapisnici također korisni su s ciljem utvrđivanja što se dogodilo u Kubernetes klasteru nakon određenog (primjerice, suspektnog) događaja tj. da se može odraditi tzv. „Post-Mortem“ analiza.

Revizijski zapisnici sustava Kubernetes prvi puta su predstavljeni u verziji 1.11. sustava Kubernetes.

Kubernetes API središte je svih događaja unutar sustava Kubernetes. Revizijski zapisnici strukturirani su u *JSON* formatu, a svaki zapisnik sadrži vrlo detaljne metapodatke. Na primjer, uključuje traženi URL put (eng. Path), HTTP metodu i podatke o korisniku koji šalje zahtjev.

Kubernetes API poslužitelj obrađuje sve promjene stanja sustava Kubernetes i pohranjuje isto na mjesto za pohranu, a prema konfiguraciji revizijskih zapisnika. Revizija omogućuje administratorima Kubernetes klastera odgovore na sljedeća pitanja:

- Što se dogodilo?
- Kada se dogodilo?
- Tko je inicirao događaj?
- Na čemu se dogodilo?
- Gdje je opaženo?
- Od kuda je pokrenut događaj?

Drugim riječima, omogućavanje revizijskih zapisnika omogućava snimanje događaja i radnji kao što su stvaranje, ažuriranje, čitanje, brisanje i sl.

Revizijski zapisi započinju svoj životni ciklus unutar *kube-apiserver* komponente Kubernetes klastera. Svaki zahtjev u svakoj fazi njegovog izvršenja generira revizijski događaj, koji se zatim prethodno obrađuje prema određenoj politici i zapisuje u za to predviđeno mjesto, a prema konfiguraciji revizijskih zapisa.

Revizijska politika određuje što se bilježi, a revizijska pozadina (eng. Audit Backends) čuva te zapise. Svaki zahtjev može se snimiti s pripadajućom fazom. Definirane faze su:

RequestReceived **-** Stadij za događaje koji se generiraju čim rukovatelj revizije (eng. Audit Handler) zaprimi zahtjev, a prije no što se delegira niz rukovatelja lanca (eng. Handler Chain),

ResponseStarted - Nakon slanja zaglavlja odgovora, ali prije slanja tijela odgovora. Ova se faza generira samo za dugotrajne zahtjeve,

ResponseComplete - Tijelo odgovora je dovršeno i više neće biti generirani bilo kakovi podaci,

Panic - događaji koji se generiraju kada se dogodi „panika“.

No, ne odnosi se svaka faza na sve zahtjeve. Osim toga, moguće je definirati razinu revizije kako bi informirali sustav o razini događaja koji se mora zabilježiti.

Razine revizije dijele se u četiri osnovne kategorije:

1. ***None*** – Sustav ne bilježi događaje koji sadrže ovaj string,
2. ***Metadata*** – sustav bilježi metapodatke zahtjeva, kao što su resurs i vremenska oznaka, ali preskače bilježenje tijela zahtjeva ili odgovora,
3. ***Request*** – Ne primjenjuje se na zahtjeve koji nisu resursi i bilježi samo metapodatke događaja i tijelo zahtjeva. Preskače tijelo odgovora,
4. ***RequestResponse*** – bilježi cijeli događaj, uključujući njegove metapodatke, zahtjeve i tijela odgovora. Ali ova naredba nije primjenjiva za zahtjeve koji nisu resursi. Moramo znati da ova kategorija generira najviše podataka!

U politici revizije (eng. Audit Policy) može se konfigurirati više pravila revizije. Svako će pravilo revizije konfigurirano je sa sljedećim poljima:

- ***level:*** Razina revizije koja definira opširnost događaja revizije,
- ***resources:*** Kubernetes objekti pod revizijom. Resursi se mogu specificirati grupom aplikacijskog programskog sučelja (API) i tipom objekta,
- ***nonResourcesURL:*** URL (eng. Uniform Resource Locator) koja nije povezan s resursima pod revizijom,
- ***namespace:*** Odlučuje koji Kubernetes objekti iz kojih imenskih prostora (eng. Namespaces) će biti pod revizijom. Prazan niz će se koristiti za odabir objekata bez prostora imena, a prazan popis podrazumijeva sve prostore imena,
- ***verb:*** Odlučuje o specifičnoj operaciji Kubernetes objekata koji će biti pod revizijom npr. stvaranje, ažuriranje ili brisanje i sl.
- ***users:*** Određuje autentificiranog korisnika na kojeg se primjenjuje pravilo revizije,
- ***userGroups:*** Određuje autentificiranu korisničku skupinu na koju se primjenjuje pravilo revizije,
- ***omitStages:*** Preskače generiranje događaja na danim fazama. Ovo se također može postaviti na razini pravila.

Revizijske pozadine (eng. Audit Backends)

Revizijske pozadine održavaju i šalju revizijske događaje u vanjsku pohranu. Prema početnim postavkama, *kube-apiserver* nudi dvije vrste pozadina revizije:

1. **Log Backend** - zapisuje događaje u datotečni sustav (eng. File System) hosta,
2. **Webhook Backend** - šalje događaje vanjskom HTTP API-ju (vanjski sustav za prikupljanje i analizu revizijskih zapisa kao što je sustav *ElasticSearch* ili sustav *Falco* (https://falco.org/).

Log Backend

Log Backend zapisuje događaje revizije u datoteku u *JSON* formatu Može se konfigurirati pozadina revizije dnevnika pomoću sljedećih *kube-apiserver* oznaka:

- ***--audit-log-path*** - navodi stazu (eng. Path) datoteke dnevnika koju pozadina dnevnika koristi za pisanje događaja revizije. Ne navođenje ove oznake onemogućuje Log backend,
- ***--audit-log-maxage*** - definira maksimalni broj dana za zadržavanje starih datoteka revizijskog dnevnika,
- ***--audit-log-maxbackup*** - definira najveći broj datoteka dnevnika revizije koje treba zadržati,
- ***--audit-log-maxsize*** - definira maksimalnu veličinu u megabajtima datoteke revizijskog dnevnika prije nego što se on rotira.

Primjer revizijske politike (*Audit Policy*):

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
# Log changes to Namespaces at the RequestResponse level.
- level: RequestResponse
  resources:
  - group: ""
    resources: ["namespaces"]
# Log pod changes in the audit-test Namespace at Request level
- level: Request
  resources:
  - group: ""
    resources: ["pods"]
  namespaces: ["test"]
# Log all ConfigMap and Secret changes at the Metadata level.
- level: Metadata
  resources:
  - group: ""
    resources: ["secrets", "configmaps"]
# Catch-all - Log all requests at the metadata level.
- level: Metadata
```

Primjer konfiguracije *--audit-policy-file, --audit-log-path, volumeMounts* i *volumes* revizijske politike (*Audit Policy*):

```
-spec:
   containers:
   - command:
     --audit-policy-file=/etc/kubernetes/audit-politika.yaml
     --audit-log-path=/var/log/kubernetes/audit/audit.log
...
volumeMounts:
  - mountPath: /etc/kubernetes/audit-politika.yaml
    name: audit
    readOnly: true
  - mountPath: /var/log/kubernetes/audit/
    name: audit-log
    readOnly: false
...
volumes:
- name: audit
  hostPath:
    path: /etc/kubernetes/audit-politika.yaml
    type: File
- name: audit-log
  hostPath:
    path: /var/log/kubernetes/audit/
    type: DirectoryOrCreate
```

Konfiguracijske postavke i atributi *--audit-policy-file, --audit-log-path, volumeMounts i volumes* revizijske politike konfiguriraju se tj. upisuju u manifest datoteku *kube-apiserver.yaml.*

Webhook backend

Webhook backend šalje revizijske događaje udaljenom web API-ju. Prilikom konfiguracije može se postaviti *Webhook* revizorski backend na način da se postave kube-apiserver *flags* kao što su::

- --*audit-webhook-config-file* - navodi stazu do datoteke s konfiguracijom webhooka. Konfiguracija webhooka zapravo je specijalizirana YAML *kubeconfig* datoteka.
- --*audit-webhook-initial-backoff* - navodi koliko vremena treba čekati nakon prvog neuspjelog zahtjeva prije ponovnog pokušaja. Sljedeći zahtjevi pokušavaju se ponovno s eksponencijalnim odmakom.

Konfiguracijska datoteka *webhook* koristi format *kubeconfig* za određivanje udaljene adrese usluge i vjerodajnica koje se koriste za povezivanje s uslugom. Pod uslugom podrazumijevamo sustave kao što su *Elasticsearch* ili *Falco*.

U slijedećem poglavlju, **7. Ažuriranja i nadogradnje sustava Kubernetes** opisati će se periodična ažuriranja i nadogradnje Kubernetes klastera kao jednog od sigurnosnih aspekata sustava Kubernetes s posebnim osvrtom na identifikaciju javno poznatih sigurnosnih ranjivosti i izloženosti sustava (eng. Common Vulnerabilities and Exposures; CVE) koji se odnose na sustav Kubernetes.

7 AŽURIRANJA I NADOGRADNJE SUSTAVA KUBERNETES

Periodična ažuriranja i nadogradnja Kubernetes klastera je proces koji ima za cilj nadograditi sustav Kubernetes s najnovijim sigurnosnim značajkama i ispravcima grešaka, kao i iskoristiti nove značajke koje se objavljuju u novim verzijama sustava.

Isto tako, nadogradnjom podižemo verziju sustava Kubernetes koja sama po sebi donosi nove značajke, ispravljene pogreške prijašnjih verzija te nove sigurnosne značajke.

Posebnu pažnju potrebno je dati ranjivostima i izloženostima (eng. Common Vulnerabilities and Exposures; CVE). CVE su identifikacije javno poznatih sigurnosnih ranjivosti i izloženosti sustava koje se nalaze u aplikacijama i sustavima kao što je sustav Kubernetes. CVE ID se sastoji od CVE niza iza kojeg slijede godina i ID broj za pojedinu ranjivost.

CVE baza podataka je javno dostupna i održava je MITER Corporation, a za sustav Kubernetes nalazi se na adresi: https://www.cvedetails.com/vulnerability-list/vendor_id-15867/product_id-34016/Kubernetes-Kubernetes.html. CVE-ovi uključuju kratak opis svakog problema, što je korisno za razumijevanje temeljnog uzroka i ozbiljnosti problema. Isti ne uključuju tehničke pojedinosti o samom problemu.

CVE-ovi su korisni stručnjacima za sigurnost sustava Kubernetes za koordinaciju i određivanje prioriteta ažuriranja. Svaki CVE ima povezanu ozbiljnost. MITER koristi Common Vulnerability Scoring System (CVSS) za dodjeljivanje ocjene ozbiljnosti CVE-u. Preporuča se odmah primijeniti CVE-ove „visoke ozbiljnosti“.

Nadogradnja Kubernetes klastera, sa ciljam da se cijeli proces odvija bez zastoja aplikacija koje se na njemu odvijaju, može biti (a u većini slučajeva i jest) veoma složen i zahtjevan proces koji se sastoji od niza međusobno povezanih procesa i koraka koji se moraju odvijati u točno određenom i zadanom redosljedu. Slijedi opći primjer nadogradnje Kubernetes klastera:

Nadogradnja Upravljačkog čvora (Master Node)

1. Ažurirati *kubeadm* na Upravljačkom čvoru
2. Zaustaviti Upravljački čvor (zaustaviti Kubernetes resurse i onemogućiti *Scheduling*),
3. Generirati plan nadogradnje pomoću *kubeadm,*
4. Izvršiti nadogradnju sustava Kubernetes,
5. Ažurirati *kubectl i kubelet,*
6. Pokrenuti upravljački čvor i omogući *Scheduling.*

Nadogradnja Radnog čvora (Worker Node)

Vrlo je važno napomenuti da se svi koraci navedeni za Radni čvor ne odvijaju paralelno na više radnih čvorova. Naravno, uz pretpostavku da se Kubernetes klaster sastoji od više Radnih čvorova što je slučaj u većini produkcijskih okolina sustava Kubernetes.

1. Ažurirati *kubeadm*
2. Zaustaviti Upravljači čvor (zaustaviti Kubernetes resurse i onemogućiti *Scheduling*),
3. Izvršite nadogradnju sustava Kubernetes
4. Ažurirajte *kubectl i kubelet*
5. Pokrenuti dotični radnički čvor (omogući *Scheduling*)
6. Povjeriti stanje Kubernetes klastera i komunikaciju Master - Worker

U slijedećem poglavlju, **8. Kube Bench sigurnosno testiranje** opisati će se Kube Bench, aplikaciju otvorenog koda, sigurnosno testiranje kao standardni sigurnosni operativni postupak, kako ga postaviti, koji su preduvjeti te na koji način interpretirati dobivene podatke i informacije o sigurnosnom stanju Kubernetes klastera.

U istom poglavlju prikazati će se i praktični primjer Kube Bench sigurnosnog testiranja. Prije samog sigurnosnog testa opisati će se Kubernetes klaster sa svim svojim osnovnim dijelovima kao i host računalo i virtualne mašine na kojem se isti nalazi.

8 KUBE BENCH SIGURNOSNO TESTIRANJE

Kube Bench je centralizirano, sveobuhvatno i referentno rješenje CIS-a (eng. Center for Internet Security) za testiranje Kubernetes klastera. Sve relevantne informacije mogu se dobiti na slijedećem linku: https://www.cisecurity.org/

Centar za internetsku sigurnost (CIS) postavio je smjernice i mjerila za siguran razvoj i održavanje softvera. Također, izrađene su smjernice i referentni testovi za osiguranje sustava Kubernetes i postizanje razine sigurnosti za Kubernetes klaster.

Smjernice CIS Kubernetes Benchmark, a odnose se na sustav Kubernetes mogu se pronaći na slijedećem linku: https://www.cisecurity.org/benchmark/kubernetes

Kube Bench je aplikacija otvorenog koda pisana u „Go“ programskom jeziku koja pokreće CIS Kubernetes Benchmark testove na klasteru kako bi osigurala određena razina sigurnosti koja zadovoljava CIS smjernice za sigurnost.

Slika 31. Kube Bench logo (Izvor: https://github.com/aquasecurity/kube-bench)

Kube Bench je vrlo koristan alat, bez obzira je li sustav Kubernetes postavljen lokalno u podatkovnom centru tj. na virtualnim strojevima ili na Cloud Kubernetes klasterima, poput AKS, EKS ili GKE.

Preduvjeti za pokretanje Kube Bench testova na Kubernetes klasteru:

- Kubernetes klaster mora biti pokrenut sa svim osnovnim modulima,
- Potrebno je imati administratorski pristup Kubernetes klasteru,
- Pristup svim master čvorovima (ukoliko ih ima više od jednog) Kubernetes klastera za izvođenje master testova,
- Pristup svim radnim čvorovima (ukoliko ih ima više od jednog) Kubernetes klastera za pokretanje testova na radnim čvorovima.

Kube Bench pruža vrlo snažan način provjere sigurnosnih postavki Kubernetes klastera. Drugim riječima, pruža mogućnost provjere jesu li ispravno postavljene i primjereno osigurane sve sigurnosne postavke Kubernetes klastera i njegovih sastavnih dijelova, modula, i sl..

Budući da je to aplikacija pisana u „*Go*“ programskom jeziku, Kube Bench je brza i daje brze, vrlo čitljive i razumljive rezultate. To je alat koji je vrlo poželjno i korisno imati ukoliko koristimo Kubernetes klaster u produkcijskoj okolini.

Najbolji dio je taj što se standard revidira s vremena na vrijeme, i s te strane Kube Bench postaje imperativ kao standardni sigurnosni operativni postupak svakog sistemskog administratora sustava Kubernetes unutar Kubernetes klastera u produkcijskom okruženju.

Kube Bench sigurnosno testiranje - praktični primjer

Prije samog prikaza praktičnog primjera sigurnosnog testiranja Kubernetes klastera potrebno je opisati kontekst/okolinu tj. opisati fizički host na kojem su postavljene virtualne mašine, virtualne mašine te Kubernetes klaster nad kojime će se odraditi sigurnosno testiranje pomoću sustava Kube Bench i Kubescape.

Za pokretanje virtualnih mašina (VM) korišten je KVM. Kernel-based Virtual Machine (KVM) je tehnologija virtualizacije otvorenog koda ugrađena u sustav Linux. KVM omogućuje pretvaranje Linux stroja u hipervizor koji omogućuje glavnom računalu (host) pokretanje više izoliranih virtualnih okruženja koja se nazivaju gosti (eng. Guests) ili virtualni mašine/strojevi (VM). KVM dolazi dio Linux distribucije Red Hat i kao takav korišten je u praktičnom dijelu ovog specijalističkog rada.

OPIS OKOLINE PRAKTIČNOG DIJELA SPECIJALISTIČKOG RADA

Fizičko računalo (PC radna stanica) na kojem su podignute virtualne mašine i na kojem je postavljen Kubernetes klaster

```
[root@kube-lab-host ~]# lshw -short
H/W path       Device     Class          Description
=====================================================
system                    10MNS3AE00 (LENOVO_MT_10MNBU_Think_FM_ThinkCentre M910t)
/0                        bus            3106
/0/0                      memory         64KiB BIOS
/0/3e                     memory         32GiB System Memory
/0/3e/0                   memory         16GiB DIMM DDR4 Synchronous Unbuffered
(Unregistered) 2667 MHz (0.4 ns)
/0/3e/1                   memory         16GiB DIMM DDR4 Synchronous Unbuffered
(Unregistered) 2667 MHz (0.4 ns)
/0/3e/2                   memory         [empty]
/0/3e/3                   memory         [empty]
/0/45                     memory         256KiB L1 cache
/0/46                     memory         1MiB L2 cache
/0/47                     memory         8MiB L3 cache
/0/48                     processor      Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz
/0/100                    bridge         Xeon E3-1200 v6/7th Gen Core Processor
Host Bridge/DRAM Registers
/0/100/2       /dev/fb0   display        HD Graphics 630

...
```

Operativni sustav na fizičkom računalu na kojem je postavljen Kubernetes klaster

```
[root@kube-lab-host ~]# cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="8.6 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
VERSION_ID="8.6"
PLATFORM_ID="platform:el8"
PRETTY_NAME="Red Hat Enterprise Linux 8.6 (Ootpa)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:redhat:enterprise_linux:8::baseos"
HOME_URL=https://www.redhat.com/
DOCUMENTATION_URL=https://access.redhat.com/documentation/red_hat_enterprise_linux/8/
BUG_REPORT_URL=https://bugzilla.redhat.com/
REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 8"
REDHAT_BUGZILLA_PRODUCT_VERSION=8.6
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"
REDHAT_SUPPORT_PRODUCT_VERSION="8.6"
```

```
[root@kube-lab-host ~]# uname -a
Linux kube-lab-host 4.18.0-372.9.1.el8.x86_64 #1 SMP Fri Apr 15 22:12:19 EDT 2022 x86_64 x86_64 x86_64 GNU/Linux
```

Popis virtualnih mašina koje sačinjavaju Kubernetes klaster

```
[root@kube-lab-host ~]# virsh list --all
 Id   Name              State
--------------------------------
 12   kube-master       running
 17   kube-node1        running
 19   kube-node2        running
```

Opis virtualnih mašina koje sačinjavaju Kubernetes klaster

```
[root@kube-lab-host ~]# virsh dominfo kube-master
Id:             12
Name:           kube-master
UUID:           07ee837e-0040-466a-b3a3-a8ed94e7adcd
OS Type:        hvm
State:          running
CPU(s):         2
CPU time:       2663743.3s
Max memory:     8388608 KiB
Used memory:    8388608 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c195,c623 (enforcing)

[root@kube-lab-host ~]# virsh dominfo kube-node1
Id:             17
Name:           kube-node1
UUID:           326cb84e-2915-47e8-a3c6-e8f6d32e919a
OS Type:        hvm
State:          running
CPU(s):         2
CPU time:       5458823.4s
Max memory:     4194304 KiB
Used memory:    4194304 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c149,c267 (enforcing)

[root@kube-lab-host ~]# virsh dominfo kube-node2
Id:             19
Name:           kube-node2
UUID:           7d7f19a3-7f22-4f0b-8925-2e345c3cde53
OS Type:        hvm
State:          running
CPU(s):         2
CPU time:       9126038.7s
Max memory:     4194304 KiB
Used memory:    4194304 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c454,c853 (enforcing)
```

Opis Kubernetes klastera:

- Jedan Master čvor (kube-master) i dva Radna čvora (kube-node1 i kube-node2)
- OS na klaster čvorovima: CentOS Linux 8 4.18.0-348.7.1.el8_5.x86_64
- Instalirana Ver. sustava Kubernetes 1.26.0
- Container Runntime: containerd Ver. 1.6.14

```
[tgolubic@kube-master ~]$ kubectl cluster-info

Kubernetes control plane is running at https://10.0.87.100:6443

CoreDNS is running at https://10.0.87.100:6443/api/v1/namespaces/kube-
system/services/kube-dns:dns/proxy

[tgolubic@kube-master ~]$ kubectl version --output=yaml

clientVersion:
  buildDate: "2022-12-08T19:58:30Z"
  compiler: gc
  gitCommit: b46a3f887ca979b1a5d14fd39cb1af43e7e5d12d
  gitTreeState: clean
  gitVersion: v1.26.0
  goVersion: go1.19.4
  major: "1"
  minor: "26"
  platform: linux/amd64
kustomizeVersion: v4.5.7
serverVersion:
  buildDate: "2022-12-08T19:51:45Z"
  compiler: gc
  gitCommit: b46a3f887ca979b1a5d14fd39cb1af43e7e5d12d
  gitTreeState: clean
  gitVersion: v1.26.0
  goVersion: go1.19.4
  major: "1"
  minor: "26"
platform: linux/amd64

[tgolubic@kube-master ~]$ kubectl config current-context

kubernetes-admin@kubernetes

[tgolubic@kube-master ~]$ kubectl version --short

Flag --short has been deprecated, and will be removed in the future.
Client Version: v1.26.0
Kustomize Version: v4.5.7
Server Version: v1.26.0
```

Kubernetes klaster čvorovi

```
[tgolubic@kube-master ~]$ kubectl get nodes -o wide
kube-master   Ready    control-plane   89d   v1.26.0   10.0.87.100   <none>
CentOS Linux 8   4.18.0-348.7.1.el8_5.x86_64   containerd://1.6.14
kube-node1    Ready    <none>          89d   v1.26.0   10.0.87.101   <none>
CentOS Linux 8   4.18.0-348.7.1.el8_5.x86_64   containerd://1.6.14
kube-node2    Ready    <none>          53d   v1.26.0   10.0.87.102   <none>
CentOS Linux 8   4.18.0-348.7.1.el8_5.x86_64   containerd://1.6.14
```

***Kube config* datoteka (kubernetes-admin) na Master čvoru**

```
[tgolubic@kube-master ~]$ kubectl config view
apiVersion: v1
clusters:
- cluster:
     certificate-authority-data: DATA+OMITTED
     server: https://10.0.87.100:6443
  name: kubernetes
contexts:
- context:
     cluster: kubernetes
     user: kubernetes-admin
     name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
   - name: kubernetes-admin
   user:
     client-certificate-data: DATA+OMITTED
     client-key-data: DATA+OMITTE
```

Popis svih *Namespace* unutar Kubernetes klastera

```
[tgolubic@kube-master ~]$ kubectl get ns -o wide
NAME                      STATUS       AGE
cattle-system             Active       60d
default                   Active       89d
elastic                   Active       49d
grafana-cloud             Active       47d
kube-bench                Active       39d
kube-node-lease           Active       89d
kube-public               Active       89d
kube-scape                Active       60d
kube-system               Active       89d
kubernetes-dashboard      Active       70d
lens-metrics              Active       69d
longhorn-storage          Active       67d
metallb-system            Active       68d
monitoring                Active       69d
prod                      Active       84d
test                      Active       85d
```

Instalacija *Kube Bench* aplikacije na Master čvor

```
# instalacija kube bench aplikacije pomoću „curl“ naredbe ili „yum“ instalera

[tgolubic@kube-master ~]$ curl -L https://github.com/aquasecurity/kube-
bench/releases/download/v0.6.2/kube-bench_0.6.2_linux_amd64.rpm -o kube-
bench_0.6.2_linux_amd64.rpm

[tgolubic@kube-master ~]$ sudo yum install kube-bench_0.6.2_linux_amd64.rpm -y

# ili ručno preuzeti i ekstrahirati kube-bench binary:

[tgolubic@kube-master ~]$ curl -L https://github.com/aquasecurity/kube-
bench/releases/download/v0.6.2/kube-bench_0.6.2_linux_amd64.tar.gz -o kube-
bench_0.6.2_linux_amd64.tar.gz

[tgolubic@kube-master ~]$ tar -xvf kube-bench_0.6.2_linux_amd64.tar.gz

# pokretanje kube bench aplikacije

[tgolubic@kube-master ~]$ kube-bench
```

Rezultati *Kube Bench* testa:

Rezultati *Kube bench* testa grupirani su u nekoliko kategorija i to:

1 Master Node Security Configuration

1.1 Master Node Configuration Files

1.2 API Server

1.3 Controller Manager

1.4 Scheduler

2 Etcd Node Configuration

3 Control Plane Configuration

4 Worker Node Security Configuration

5 Kubernetes Policies

Nakon svake grupe testova *Kube Bench* aplikacija predlaže rješenja tj. sanacije (eng. Remediations) za sve pojedinačne testove sa rezultatima **[WARN]** i **[FAIL].** Isto tako, na kraju svake grupe testova nalazi se i zbirni prikaz svih testova.

1 Master Node Security Configuration

```
[INFO] 1 Master Node Security Configuration

[INFO] 1.1 Master Node Configuration Files
[PASS] 1.1.1 Ensure that the API server pod specification file permissions are set
to 644 or more restrictive (Automated)
[PASS] 1.1.2 Ensure that the API server pod specification file ownership is set to
root:root (Automated)
[PASS] 1.1.3 Ensure that the controller manager pod specification file permissions
are set to 644 or more restrictive (Automated)
[PASS] 1.1.4 Ensure that the controller manager pod specification file ownership
is set to root:root (Automated)
[PASS] 1.1.5 Ensure that the scheduler pod specification file permissions are set
to 644 or more restrictive (Automated)
[PASS] 1.1.6 Ensure that the scheduler pod specification file ownership is set to
root:root (Automated)
[PASS] 1.1.7 Ensure that the etcd pod specification file permissions are set to
644 or more restrictive (Automated)
[PASS] 1.1.8 Ensure that the etcd pod specification file ownership is set to
root:root (Automated)
[WARN] 1.1.9 Ensure that the Container Network Interface file permissions are set
to 644 or more restrictive (Manual)
[WARN] 1.1.10 Ensure that the Container Network Interface file ownership is set to
root:root (Manual)
[PASS] 1.1.11 Ensure that the etcd data directory permissions are set to 700 or
more restrictive (Automated)
[FAIL] 1.1.12 Ensure that the etcd data directory ownership is set to etcd:etcd
(Automated)
[PASS] 1.1.13 Ensure that the admin.conf file permissions are set to 644 or more
restrictive (Automated)
[PASS] 1.1.14 Ensure that the admin.conf file ownership is set to root:root
(Automated)
[PASS] 1.1.15 Ensure that the scheduler.conf file permissions are set to 644 or
more restrictive (Automated)
[PASS] 1.1.16 Ensure that the scheduler.conf file ownership is set to root:root
(Automated)
[PASS] 1.1.17 Ensure that the controller-manager.conf file permissions are set to
644 or more restrictive (Automated)
[PASS] 1.1.18 Ensure that the controller-manager.conf file ownership is set to
root:root (Automated)
[PASS] 1.1.19 Ensure that the Kubernetes PKI directory and file ownership is set
to root:root (Automated)
[PASS] 1.1.20 Ensure that the Kubernetes PKI certificate file permissions are set
to 644 or more restrictive (Manual)
[PASS] 1.1.21 Ensure that the Kubernetes PKI key file permissions are set to 600
(Manual)
```

1.2 API Server

```
[INFO] 1.2 API Server
[WARN] 1.2.1 Ensure that the --anonymous-auth argument is set to false (Manual)
[PASS] 1.2.2 Ensure that the --basic-auth-file argument is not set (Automated)
[PASS] 1.2.3 Ensure that the --token-auth-file parameter is not set (Automated)
[PASS] 1.2.4 Ensure that the --kubelet-https argument is set to true (Automated)
[PASS] 1.2.5 Ensure that the --kubelet-client-certificate and --kubelet-client-key
arguments are set as appropriate (Automated)
[FAIL] 1.2.6 Ensure that the --kubelet-certificate-authority argument is set as
appropriate (Automated)
[PASS] 1.2.7 Ensure that the --authorization-mode argument is not set to
AlwaysAllow (Automated)
[PASS] 1.2.8 Ensure that the --authorization-mode argument includes Node
(Automated)
[PASS] 1.2.9 Ensure that the --authorization-mode argument includes RBAC
(Automated)
[WARN] 1.2.10 Ensure that the admission control plugin EventRateLimit is set
(Manual)
[PASS] 1.2.11 Ensure that the admission control plugin AlwaysAdmit is not set
(Automated)
[WARN] 1.2.12 Ensure that the admission control plugin AlwaysPullImages is set
(Manual)
[WARN] 1.2.13 Ensure that the admission control plugin SecurityContextDeny is set
if PodSecurityPolicy is not used (Manual)
[PASS] 1.2.14 Ensure that the admission control plugin ServiceAccount is set
(Automated)
[PASS] 1.2.15 Ensure that the admission control plugin NamespaceLifecycle is set
(Automated)
[FAIL] 1.2.16 Ensure that the admission control plugin PodSecurityPolicy is set
(Automated)
[PASS] 1.2.17 Ensure that the admission control plugin NodeRestriction is set
(Automated)
[PASS] 1.2.18 Ensure that the --insecure-bind-address argument is not set
(Automated)
[FAIL] 1.2.19 Ensure that the --insecure-port argument is set to 0 (Automated)
[PASS] 1.2.20 Ensure that the --secure-port argument is not set to 0 (Automated)
[FAIL] 1.2.21 Ensure that the --profiling argument is set to false (Automated)
[FAIL] 1.2.22 Ensure that the --audit-log-path argument is set (Automated)
[FAIL] 1.2.23 Ensure that the --audit-log-maxage argument is set to 30 or as
appropriate (Automated)
[FAIL] 1.2.24 Ensure that the --audit-log-maxbackup argument is set to 10 or as
appropriate (Automated)
[FAIL] 1.2.25 Ensure that the --audit-log-maxsize argument is set to 100 or as
appropriate (Automated)
[WARN] 1.2.26 Ensure that the --request-timeout argument is set as appropriate
(Automated)
[PASS] 1.2.27 Ensure that the --service-account-lookup argument is set to true
(Automated)

...
```

1.3 Controller Manager

```
[INFO] 1.3 Controller Manager
[WARN] 1.3.1 Ensure that the --terminated-pod-gc-threshold argument is set as
appropriate (Manual)
[FAIL] 1.3.2 Ensure that the --profiling argument is set to false (Automated)
[PASS] 1.3.3 Ensure that the --use-service-account-credentials argument is set to
true (Automated)
[PASS] 1.3.4 Ensure that the --service-account-private-key-file argument is set as
appropriate (Automated)
[PASS] 1.3.5 Ensure that the --root-ca-file argument is set as appropriate
(Automated)
[PASS] 1.3.6 Ensure that the RotateKubeletServerCertificate argument is set to
true (Automated)
[PASS] 1.3.7 Ensure that the --bind-address argument is set to 127.0.0.1
(Automated)
```

1.4 Scheduler

```
[INFO] 1.4 Scheduler
[FAIL] 1.4.1 Ensure that the --profiling argument is set to false (Automated)
[PASS] 1.4.2 Ensure that the --bind-address argument is set to 127.0.0.1
(Automated)
```

Prijedlozi rješenja i zbirni rezultat testiranja za Master čvor

```
== Remediations master ==

1.1.9 Run the below command (based on the file location on your system) on the
master node. For example, chmod 644 <path/to/cni/files>

1.1.10 Run the below command (based on the file location on your system) on the
master node. For example, chown root:root <path/to/cni/files>

1.1.12 On the etcd server node, get the etcd data directory, passed as an argument
--data-dir, from the below command: ps -ef | grep etcd Run the below command
(based on the etcd data directory found above). For example, chown etcd:etcd
/var/lib/etcd

1.2.1 Edit the API server pod specification file /etc/kubernetes/manifests/kube-
apiserver.yaml on the master node and set the below parameter. --anonymous-
auth=false

1.2.6 Follow the Kubernetes documentation and setup the TLS connection between
the apiserver and kubelets. Then, edit the API server pod specification file
/etc/kubernetes/manifests/kube-apiserver.yaml on the master node and set the
--kubelet-certificate-authority parameter to the path to the cert file for the
certificate authority. --kubelet-certificate-authority=<ca-string>

1.2.10 Follow the Kubernetes documentation and set the desired limits in a
configuration file. Then, edit the API server pod specification file
/etc/kubernetes/manifests/kube-apiserver.yaml and set the below parameters.
--enable-admission-plugins=...,EventRateLimit,... --admission-control-config-
file=<path/to/configuration/file>

1.2.12 Edit the API server pod specification file /etc/kubernetes/manifests/kube-
apiserver.yaml on the master node and set the --enable-admission-plugins parameter
to include AlwaysPullImages. --enable-admission-plugins=...,AlwaysPullImages,...

1.2.13 Edit the API server pod specification file /etc/kubernetes/manifests/kube-
apiserver.yaml on the master node and set the --enable-admission-plugins parameter
to include SecurityContextDeny, unless PodSecurityPolicy is already in place.
--enable-admission-plugins=...,SecurityContextDeny,...

1.2.16 Follow the documentation and create Pod Security Policy objects as per your
environment. Then, edit the API server pod specification file
/etc/kubernetes/manifests/kube-apiserver.yaml on the master node and set the --
enable-admission-plugins parameter to a value that includes PodSecurityPolicy:
--enable-admission-plugins=...,PodSecurityPolicy,... Then restart the API Server.

1.2.19 Edit the API server pod specification file /etc/kubernetes/manifests/kube-
apiserver.yaml on the master node and set the below parameter. --insecure-port=0

1.2.21 Edit the API server pod specification file /etc/kubernetes/manifests/kube-
apiserver.yaml on the master node and set the below parameter. --profiling=false

...


== Summary master ==

43    checks   PASS
11    checks   FAIL
11    checks   WARN
0     checks   INFO
```

2 Etcd Node Configuration

```
[INFO] 2 Etcd Node Configuration

[INFO] 2 Etcd Node Configuration Files

[PASS] 2.1 Ensure that the --cert-file and --key-file arguments are set as
appropriate (Automated)
[PASS] 2.2 Ensure that the --client-cert-auth argument is set to true (Automated)
[PASS] 2.3 Ensure that the --auto-tls argument is not set to true (Automated)
[PASS] 2.4 Ensure that the --peer-cert-file and --peer-key-file arguments are set
as appropriate (Automated)
[PASS] 2.5 Ensure that the --peer-client-cert-auth argument is set to true
(Automated)
[PASS] 2.6 Ensure that the --peer-auto-tls argument is not set to true (Automated)
[PASS] 2.7 Ensure that a unique Certificate Authority is used for etcd (Manual)


== Summary etcd ==

7     checks   PASS
0     checks   FAIL
0     checks   WARN
0     checks   INFO
```

3 Control Plane Configuration

```
[INFO] 3 Control Plane Configuration

[INFO] 3.1 Authentication and Authorization
[WARN] 3.1.1 Client certificate authentication should not be used for users
(Manual)

[INFO] 3.2 Logging
[WARN] 3.2.1 Ensure that a minimal audit policy is created (Manual)
[WARN] 3.2.2 Ensure that the audit policy covers key security concerns (Manual)

== Remediations controlplane ==

3.1.1 Alternative mechanisms provided by Kubernetes such as the use of OIDC should
be implemented in place of client certificates.

3.2.1 Create an audit policy file for your cluster.

3.2.2 Consider modification of the audit policy in use on the cluster to include
these items, at a minimum.


== Summary controlplane ==

0     checks   PASS
0     checks   FAIL
3     checks   WARN
0     checks   INFO
```

4 Worker Node Security Configuration

```
[INFO] 4 Worker Node Security Configuration

[INFO] 4.1 Worker Node Configuration Files
[PASS] 4.1.1 Ensure that the kubelet service file permissions are set to 644 or
more restrictive (Automated)
[PASS] 4.1.2 Ensure that the kubelet service file ownership is set to root:root
(Automated)
[PASS] 4.1.3 If proxy kubeconfig file exists ensure permissions are set to 644 or
more restrictive (Manual)
[PASS] 4.1.4 Ensure that the proxy kubeconfig file ownership is set to root:root
(Manual)
[PASS] 4.1.5 Ensure that the --kubeconfig kubelet.conf file permissions are set to
644 or more restrictive (Automated)
[PASS] 4.1.6 Ensure that the --kubeconfig kubelet.conf file ownership is set to
root:root (Manual)
[PASS] 4.1.7 Ensure that the certificate authorities file permissions are set to
644 or more restrictive (Manual)
[PASS] 4.1.8 Ensure that the client certificate authorities file ownership is set
to root:root (Manual)
[PASS] 4.1.9 Ensure that the kubelet --config configuration file has permissions
set to 644 or more restrictive (Automated)
[PASS] 4.1.10 Ensure that the kubelet --config configuration file ownership is set
to root:root (Automated)

[INFO] 4.2 Kubelet
[PASS] 4.2.1 Ensure that the anonymous-auth argument is set to false (Automated)
[PASS] 4.2.2 Ensure that the --authorization-mode argument is not set to
AlwaysAllow (Automated)
[PASS] 4.2.3 Ensure that the --client-ca-file argument is set as appropriate
(Automated)
[PASS] 4.2.4 Ensure that the --read-only-port argument is set to 0 (Manual)
[PASS] 4.2.5 Ensure that the --streaming-connection-idle-timeout argument is not
set to 0 (Manual)
[FAIL] 4.2.6 Ensure that the --protect-kernel-defaults argument is set to true
(Automated)
[PASS] 4.2.7 Ensure that the --make-iptables-util-chains argument is set to true
(Automated)
[PASS] 4.2.8 Ensure that the --hostname-override argument is not set (Manual)
[WARN] 4.2.9 Ensure that the --event-qps argument is set to 0 or a level which
ensures appropriate event capture (Manual)
[WARN] 4.2.10 Ensure that the --tls-cert-file and --tls-private-key-file arguments
are set as appropriate (Manual)
[PASS] 4.2.11 Ensure that the --rotate-certificates argument is not set to false
(Manual)
[PASS] 4.2.12 Verify that the RotateKubeletServerCertificate argument is set to
true (Manual)
[WARN] 4.2.13 Ensure that the Kubelet only makes use of Strong Cryptographic
Ciphers (Manual)
```

Prijedlozi rješenja i zbirni rezultat testiranja za Radne čvorove

```
== Remediations node ==

4.2.6 If using a Kubelet config file, edit the file to set protectKernelDefaults:
true. If using command line arguments, edit the kubelet service file
/lib/systemd/system/kubelet.service on each worker node and set the below
parameter in KUBELET_SYSTEM_PODS_ARGS variable.--protect-kernel-defaults=true
Based on your system, restart the kubelet service. For example: systemctl daemon-
reload systemctl restart kubelet.service

4.2.9 If using a Kubelet config file, edit the file to set eventRecordQPS: to an
appropriate level. If using command line arguments, edit the kubelet service file
/lib/systemd/system/kubelet.service on each worker node and set the below
parameter in KUBELET_SYSTEM_PODS_ARGS variable. Based on your system, restart the
kubelet service. For example: systemctl daemon-reload systemctl restart
kubelet.service

4.2.10 If using a Kubelet config file, edit the file to set tlsCertFile to the
location of the certificate file to use to identify this Kubelet, and
tlsPrivateKeyFile to the location of the corresponding private key file. If using
command line arguments, edit the kubelet service file
/lib/systemd/system/kubelet.service on each worker node and set the below
parameters in KUBELET_CERTIFICATE_ARGS variable.
--tls-cert-file=<path/to/tls-certificate-file>
--tls-private-key-file=<path/to/tls-key-file>

Based on your system, restart the kubelet service. For example: systemctl daemon-
reload systemctl restart kubelet.service

4.2.13 If using a Kubelet config file, edit the file to set TLSCipherSuites: to
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_
ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE
_RSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_A
ES_256_GCM_SHA384,TLS_RSA_WITH_AES_128_GCM_SHA256

or to a subset of these values. If using executable arguments, edit the kubelet
service file /lib/systemd/system/kubelet.service on each worker node and set the -
-tls-cipher-suites parameter as follows, or to a subset of these values.

--tls-cipher-
suites=TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA2
56,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TL
S_ECDHE_RSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_RSA
_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_128_GCM_SHA256

Based on your system, restart the kubelet service. For example: systemctl daemon-
reload systemctl restart kubelet.service


== Summary node ==

19     checks    PASS
1      checks    FAIL
3      checks    WARN
0      checks    INFO
```

5 Kubernetes Policies

```
[INFO] 5 Kubernetes Policies

[INFO] 5.1 RBAC and Service Accounts
[WARN] 5.1.1 Ensure that the cluster-admin role is only used where required
(Manual)
[WARN] 5.1.2 Minimize access to secrets (Manual)
[WARN] 5.1.3 Minimize wildcard use in Roles and ClusterRoles (Manual)
[WARN] 5.1.4 Minimize access to create pods (Manual)
[WARN] 5.1.5 Ensure that default service accounts are not actively used. (Manual)
[WARN] 5.1.6 Ensure that Service Account Tokens are only mounted where necessary
(Manual)

[INFO] 5.2 Pod Security Policies
[WARN] 5.2.1 Minimize the admission of privileged containers (Manual)
[WARN] 5.2.2 Minimize the admission of containers wishing to share the host
process ID namespace (Manual)
[WARN] 5.2.3 Minimize the admission of containers wishing to share the host IPC
namespace (Manual)
[WARN] 5.2.4 Minimize the admission of containers wishing to share the host
network namespace (Manual)
[WARN] 5.2.5 Minimize the admission of containers with allowPrivilegeEscalation
(Manual)
[WARN] 5.2.6 Minimize the admission of root containers (Manual)
[WARN] 5.2.7 Minimize the admission of containers with the NET_RAW capability
(Manual)
[WARN] 5.2.8 Minimize the admission of containers with added capabilities (Manual)
[WARN] 5.2.9 Minimize the admission of containers with capabilities assigned
(Manual)

[INFO] 5.3 Network Policies and CNI
[WARN] 5.3.1 Ensure that the CNI in use supports Network Policies (Manual)
[WARN] 5.3.2 Ensure that all Namespaces have Network Policies defined (Manual)

[INFO] 5.4 Secrets Management
[WARN] 5.4.1 Prefer using secrets as files over secrets as environment variables
(Manual)
[WARN] 5.4.2 Consider external secret storage (Manual)

[INFO] 5.5 Extensible Admission Control
[WARN] 5.5.1 Configure Image Provenance using ImagePolicyWebhook admission
controller (Manual)

[INFO] 5.6 General Policies
[WARN] 5.6.1 Create administrative boundaries between resources using namespaces
(Manual)
[WARN] 5.6.2 Ensure that the seccomp profile is set to docker/default in your pod
definitions (Manual)
[WARN] 5.6.3 Apply Security Context to Your Pods and Containers (Manual)
[WARN] 5.6.4 The default namespace should not be used (Manual)
```

Prijedlozi rješenja i zbirni rezultat testiranja za Kubernetes Policies

```
== Remediations policies ==

5.1.1 Identify all clusterrolebindings to the cluster-admin role. Check if they
are used and if they need this role or if they could use a role with fewer
privileges. Where possible, first bind users to a lower privileged role and then
remove the clusterrolebinding to the cluster-admin role: kubectl delete
clusterrolebinding [name]

5.1.2 Where possible, remove get, list and watch access to secret objects in the
cluster.

5.1.3 Where possible replace any use of wildcards in clusterroles and roles with
specific objects or actions.

5.1.4 Where possible, remove create access to pod objects in the cluster.

5.1.5 Create explicit service accounts wherever a Kubernetes workload requires
specific access to the Kubernetes API server. Modify the configuration of each
default service account to include this value automountServiceAccountToken: false

5.1.6 Modify the definition of pods and service accounts which do not need to
mount service account tokens to disable it.

5.2.1 Create a PSP as described in the Kubernetes documentation, ensuring that
the .spec.privileged field is omitted or set to false.

5.2.2 Create a PSP as described in the Kubernetes documentation, ensuring that the
.spec.hostPID field is omitted or set to false.

5.2.3 Create a PSP as described in the Kubernetes documentation, ensuring that the
.spec.hostIPC field is omitted or set to false.

5.2.4 Create a PSP as described in the Kubernetes documentation, ensuring that the
.spec.hostNetwork field is omitted or set to false.

5.2.5 Create a PSP as described in the Kubernetes documentation, ensuring that the
.spec.allowPrivilegeEscalation field is omitted or set to false.

5.2.6 Create a PSP as described in the Kubernetes documentation, ensuring that the
.spec.runAsUser.rule is set to either MustRunAsNonRoot or MustRunAs with the range
of UIDs not including 0.

5.2.7 Create a PSP as described in the Kubernetes documentation, ensuring that the
.spec.requiredDropCapabilities is set to include either NET_RAW or ALL.

5.2.8 Ensure that allowedCapabilities is not present in PSPs for the cluster
unless it is set to an empty array.

...


== Summary policies ==

0     checks   PASS
0     checks   FAIL
24    checks   WARN

0     checks   INFO
```

Zbirni rezultati za sve sigurnosne testove Kube Bench aplikacije

```
== Summary total ==
69 checks PASS
12 checks FAIL
41 checks WARN
0 checks INFO
```

U slijedećem poglavlju, **9. Kubescape sigurnosno testiranje** opisati će se alat, odnosno aplikacija otvorenog koda Kubescape koja služi za sigurnosna testiranja Kubernetes klastera, *YAML* konfiguracijskih datoteka i *HELM* chart-ova. U praktičnom dijelu poglavlja opisati će se i prikazati sigurnosno testiranje pomoću Kubescape aplikacije.

9 KUBESCAPE SIGURNOSNO TESTIRANJE

Kubescape je K8s alat/aplikacija otvorenog koda koji timovima zaduženim za sigurnost sustava Kubernetes pruža mogućnosti kao što su analiza rizika, sigurnosnu usklađenost, RBAC vizualizator i skeniranje ranjivosti slika spremnika (eng. Images).

Više informacija o *Kubescape* aplikaciji za sigurnosna skeniranja može se naći na slijedećem linku: https://www.armosec.io/kubescape/.

Kubescape skenira Kubernetes klastere, *YAML* konfiguracijske datoteke i *HELM* chart-ove (https://helm.sh/) , otkrivajući pogrešne konfiguracije prema smjernicama i mjerilima kao što su NSA (https://www.nsa.gov/), MITRE ATT&CK (https://attack.mitre.org/), softverske ranjivosti i *RBAC* (kontrola pristupa temeljena na ulogama), kršenja u ranim fazama CI/CD i sl.

Kubescape alat omogućava skeniranje na nivou yaml datoteke (np. pojedinih *Deployment* i sl.), na nivou *Namespace* i sl. Rezultate pojedinih sigurnosnih skeniranja moguće je pohraniti i .pdf ili .html datoteku.

Kubescape ima mogućnost trenutnog izračunava ocjenu rizika i prikazuje trendove rizika.

Slika 32. Kubescape logo (Izvor: https://github.com/kubescape/kubescape)

Kubescape je postao jedan od najbrže rastućih Kubernetes alata za sigurnosna testiranja među razvojnim programerima zahvaljujući svom CLI sučelju jednostavnom za korištenje, fleksibilnim izlaznim formatima i mogućnostima automatskog skeniranja, štedeći korisnicima i administratorima sustava Kubernetes dragocjeno vrijeme, trud i resurse.

Kubescape se izvorno može integrirati s drugim DevOps alatima, uključujući Jenkins, CircleCI, GitHub, GitLab, Prometheus i Slack, te podržava cloud Kubernetes implementacije kao što su EKS, GKE i AKS.

Kako analizirati rezultate Kubescape skeniranja

Analiza rizika/sigurnosnog skeniranja ima sljedeća polja:

(1) Ozbiljnost prijetnje: Ovo polje mjeri koliko je prijetnja kritična. Postoje tri razine prijetnje, a to su: visoka, srednja i niska,
(2) Naziv kontrole: ovo polje navodi naziv komponente ili aspekta koji se analizira,
(3) Dokumenti: ovo polje sadrži vezu koja služi za preusmjeravanje na stranicu koja sadrži informacije o riziku i prijetnji otkrivenoj tijekom skeniranja,
(4) Prijedlog ispravka: ovo polje sadrži promjene koje se mogu izvršiti u npr. YAML datoteci ili klasteru kako bi se prijetnja uklonila.

Ukoliko se koristi oznaka *--verbose*, prikazati će se veći broj redaka, što može dovesti do, ukoliko koristimo CLI, situacije u kojoj ćemo vrlo teško, zbog količine podataka, analizirati ono što je važno. Da bi sortirali dobivene podatke, iste možemo sortirali i spremiti/ispisati rezultat u PDF formatu.

To možemo postići dodavanjem *--format pdf --output filename.pdf* u naredbu skeniranja kao naprimjer: *# kubescape scan --format pdf --output rezultat-skeniranja.pdf*

Kubescape sigurnosno testiranje - praktični primjer

Okolina nad kojim je odrađeno sigurnosno testiranje pomoću Kubescape aplikacije identično je onom iz prethodnog poglavlja „**8 Kube Bench sigurnosno testiranje“**, (str. 144 - 149) te nije potrebno još jednom opisivati host, VM-ove, Kubernetes klaster i sl.

Proces instalacije Kubescape aplikacije na Kubernetes Master čvoru

```
[tgolubic@kube-master ~]$ curl -s
https://raw.githubusercontent.com/kubescape/kubescape/master/install.sh |
/bin/bash

Installing Kubescape...

######################################################################## 100.0%

Old installation found at /home/tgolubic/.local/bin/kubescape, do you want to
remove it? [y/n]:

Removed old installation at /home/tgolubic/.local/bin/kubescape

Finished Installation.

Your current version is: v2.2.6 [git enabled in build: true]

Usage: $ kubescape scan --enable-host-scan

Remember to add the Kubescape CLI to your path with:

  export PATH=$PATH:/home/tgolubic/.kubescape/bin


[tgolubic@kube-master ~]$ kubescape version

Your current version is: v2.2.6 [git enabled in build: true]


# slijedeće naredbe odnose se na --help tj. pomoć pri korištenju Kubescape

[tgolubic@kube-master ~]$ kubescape -h

[tgolubic@kube-master ~]$ kubescape scan -h
```

Postavljanje host skener-a na Kubernetes čvorove i sken Kubernetes klastera

```
[tgolubic@kube-master ~]$ kubescape scan --enable-host-scan
[info] Kubescape scanner starting
[info] Installing host scanner
[info] Downloading/Loading policy definitions
[success] Downloaded/Loaded policy
[info] Accessing Kubernetes objects
[success] Accessed to Kubernetes objects
[info] Requesting images vulnerabilities results
[success] Requested images vulnerabilities results
[info] Requesting Host scanner data
[info] Host scanner version : v1.0.54
[success] Requested Host scanner data
[info] Scanning. Cluster: kubernetes-admin-kubernetes
Control: C-0014 100% |
[success] Done scanning. Cluster: kubernetes-admin-kubernetes
```

Kubescape skeniranje prema NSA framework

```
[tgolubic@kube-master ~]$ kubescape scan framework nsa
[info] Kubescape scanner starting
[info] Downloading/Loading policy definitions
[success] Downloaded/Loaded policy
[info] Accessing Kubernetes objects
[success] Accessed to Kubernetes objects
[info] Scanning. Cluster: kubernetes-admin-kubernetes
Control: C-0030 100%
[success] Done scanning. Cluster: kubernetes-admin-kubernetes
```

SEVERITY	CONTROL NAME	FAILED RESOURCES	ALL RESOURCES	% RISK-SCORE
Critical	Disable anonymous access to Kubelet service	0	0	Action Required *
Critical	Enforce Kubelet client TLS authentication	0	0	Action Required *
High	Resource limits	31	39	85%
High	Applications credentials in configuration files	0	74	Action Required **
High	Host PID/IPC privileges	3	39	5%
High	HostNetwork access	5	39	8%
High	Insecure capabilities	2	39	3%
High	Privileged container	8	39	13%
Medium	Exec into container	3	98	3%
Medium	Non-root containers	32	39	87%
Medium	Allow privilege escalation	31	39	85%
Medium	Ingress and Egress blocked	33	39	88%
Medium	Automatic mapping of service account	61	105	65%
Medium	Cluster-admin binding	3	98	3%
Medium	Container hostPort	1	39	2%
Medium	Cluster internal networking	12	16	75%
Medium	Linux hardening	31	39	85%
Medium	Secret/ETCD encryption enabled	1	1	100%
Medium	Audit logs enabled	1	1	100%
Low	Immutable container filesystem	31	39	85%
Low	PSP enabled	1	1	100%
	RESOURCE SUMMARY	77	254	36.18%

FRAMEWORK NSA

* enable-host-scan flag not used. For more information: https://hub.armosec.io/docs/host-sensor
** Control configurations are empty
[info] Run with '--verbose'/'-v' flag for detailed resources view

Slika 33. Rezultat Kubescape sigurnosnog skeniranja prema NSA framework

Kubescape skeniranje prema MITRE framework

```
[tgolubic@kube-master ~]$ kubescape scan framework mitre
[info] Kubescape scanner starting
[info] Downloading/Loading policy definitions
[success] Downloaded/Loaded policy
[info] Accessing Kubernetes objects
[success] Accessed to Kubernetes objects
[info] Scanning. Cluster: kubernetes-admin-kubernetes
Control: C-0053 100%
[success] Done scanning. Cluster: kubernetes-admin-kubernetes
```

SEVERITY	CONTROL NAME	FAILED RESOURCES	ALL RESOURCES	% RISK-SCORE
Critical	Disable anonymous access to Kubelet service	0	0	Action Required *
Critical	Enforce Kubelet client TLS authentication	0	0	Action Required *
High	Applications credentials in configuration files	0	74	Action Required **
High	List Kubernetes secrets	9	98	9%
High	Writable hostPath mount	14	39	37%
High	HostPath mount	14	39	37%
High	Privileged container	8	39	13%
Medium	Exec into container	3	98	3%
Medium	Data Destruction	6	98	6%
Medium	Delete Kubernetes events	4	98	4%
Medium	Cluster-admin binding	3	98	3%
Medium	CoreDNS poisoning	4	98	4%
Medium	Malicious admission controller (mutating)	1	1	100%
Medium	Access container service account	34	71	48%
Medium	Cluster internal networking	12	16	75%
Medium	Secret/ETCD encryption enabled	1	1	100%
Medium	Audit logs enabled	1	1	100%
Low	Malicious admission controller (validating)	2	2	100%
Low	PSP enabled	1	1	100%
	RESOURCE SUMMARY	67	208	13.26%

```
FRAMEWORK MITRE

* enable-host-scan flag not used. For more information: https://hub.armosec.io/docs/host-sensor
** Control configurations are empty
[info] Run with '--verbose'/'-v' flag for detailed resources view
```

Slika 34. Rezultat Kubescape sigurnosnog skeniranja prema MITRE framework

Kubescape skeniranje objekata u *Namespace* „test“ prema MITRE i NSA framework

```
[tgolubic@kube-master ~]$ kubescape scan --include-namespaces test
[info] Kubescape scanner starting
[info] Downloading/Loading policy definitions
[success] Downloaded/Loaded policy
[info] Accessing Kubernetes objects
[success] Accessed to Kubernetes objects
[info] Requesting images vulnerabilities results
[success] Requested images vulnerabilities results
[info] Scanning. Cluster: kubernetes-admin-kubernetes
Control: C-0017 100%
[success] Done scanning. Cluster: kubernetes-admin-kubernetes
```

SEVERITY	CONTROL NAME	FAILED RESOURCES	ALL RESOURCES	% RISK-SCORE
Critical	Disable anonymous access to Kubelet service	0	0	Action Required *
Critical	Enforce Kubelet client TLS authentication	0	0	Action Required *
High	Forbidden Container Registries	0	3	Action Required ****
High	Resources memory limit and request	0	3	Action Required ****
High	Resource limits	3	3	100%
High	Applications credentials in configuration files	0	4	Action Required ****
High	List Kubernetes secrets	1	1	100%
High	Resources CPU limit and request	0	3	Action Required ****
High	Workloads with Critical vulnerabilities exposed to...	0	0	Action Required **
High	Workloads with RCE vulnerabilities exposed to exte...	0	0	Action Required **
High	RBAC enabled	0	0	Action Required ***
Medium	Exec into container	1	1	100%
Medium	Data Destruction	1	1	100%
Medium	Non-root containers	3	3	100%
Medium	Allow privilege escalation	3	3	100%
Medium	Ingress and Egress blocked	3	3	100%
Medium	Delete Kubernetes events	1	1	100%
Medium	Automatic mapping of service account	4	4	100%
Medium	Cluster-admin binding	1	1	100%
Medium	CoreDNS poisoning	1	1	100%
Medium	Cluster internal networking	1	1	100%
Medium	Linux hardening	3	3	100%
Medium	Configured liveness probe	3	3	100%
Medium	Portforwarding privileges	1	1	100%
Medium	No impersonation	1	1	100%
Medium	Secret/ETCD encryption enabled	0	0	Action Required ***
Medium	Audit logs enabled	0	0	Action Required ***
Medium	Images from allowed registry	0	3	Action Required ****
Medium	CVE-2022-0185-linux-kernel-container-escape	0	0	Action Required *
Medium	Workloads with excessive amount of vulnerabilities	0	0	Action Required **
Medium	CVE-2022-0492-cgroups-container-escape	3	3	100%
Low	Immutable container filesystem	3	3	100%
Low	Configured readiness probe	3	3	100%
Low	Network mapping	1	1	100%
Low	PSP enabled	0	0	Action Required ***
Low	Label usage for resources	2	3	81%
Low	K8s common labels usage	3	3	100%
	RESOURCE SUMMARY	6	15	36.04%

FRAMEWORKS: AllControls (risk: 35.77), NSA (risk: 49.68), MITRE (risk: 13.99)

Slika 35. Rezultat Kubescape sigurnosnog skeniranja *Namespace* „test“ prema MITRE i NSA framework

10 ZAKLJUČAK

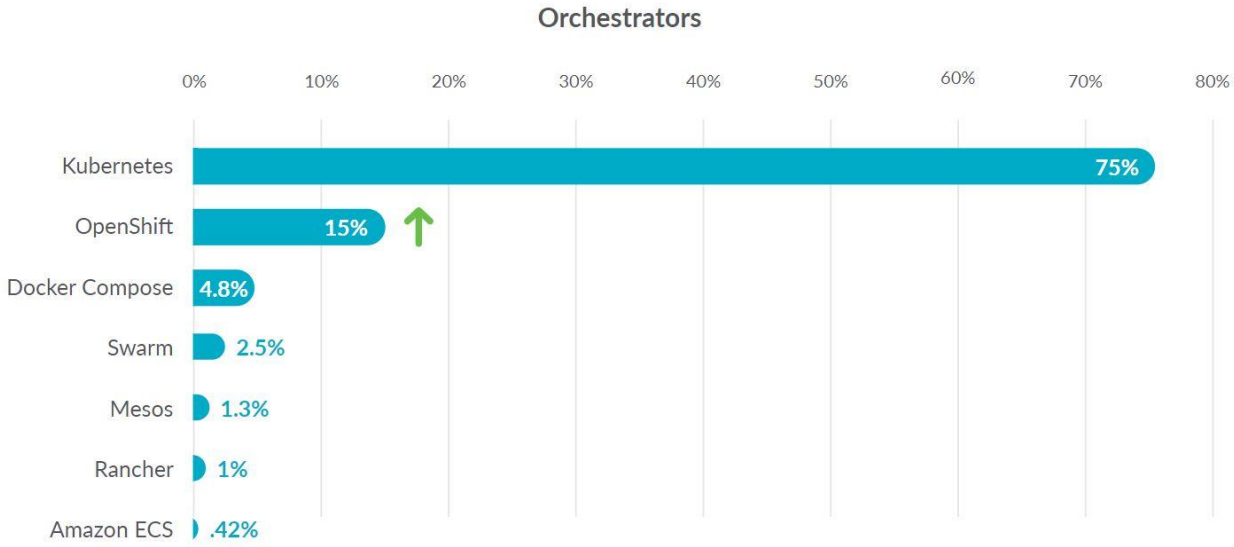
U današnje vrijeme sustav za orkestraciju Kubernetes popularniji je više nego ikada. Veliki broj organizacija koristi sustav Kubernetes za implementaciju, upravljanje i orkestraciju kontejnerima u brzorastućem Kubernetes tj. generalno u Cloud Native ekosustavu kao što se može vidjeti u CNCF Cloud Native Interactive Landscape na: https://landscape.cncf.io/.

Prema istraživanju tvrtke *Sysdig*, sustav Kubernetes u ovom trenutku ima vrlo uvjerljivo vodstvo u odnosu na druge orkestratore.

Grafički prikaz na **Slici 36.** prikazuje analizu za 2021. godinu. Za razliku od sustava Kubrnetes Swarm, Mesos i ostali orkestratori su na približno istim razinama korištenja, u usporedbi s 2020. godinom.

Red Hat OpenShift bilježi najveći skok (s 9% na 15%) u odnosu na 2020. jer, se čini da se sve više korisnika oslanja na Red Hat OpenShift, između ostaloga zbog njegove mogućnosti rada u više okruženja u isto vrijeme, ugrađenih sigurnosnih mehanizama, veoma intuitivnog grafičkog sučelja i vrlo dobre podrške za *CI/CD*.

Docker Compose, koji se koristi za upravljanje višestrukim spremnicima, ali samo na jednom glavnom hostu, dodan je u analizu za 2021. godinu iako se ne smatra orkestratorom s implementacijom na više čvorova kao što je to sustav Kubernetes.



Slika 36. 2021 Container Security and Usage Report; Container orchestration platforms (Izvor: https://dig.sysdig.com/c/pf-2021-container-security-and-usage-report?x=hJvo1P#page=1)

Alati kao što su Amazon Elastic Kubernetes Services (EKS), Google Kubernetes Engine (GKE), Amazon Kubernetes Services (AKS) kao i druge Kubernetes usluge u oblaku olakšavaju timovima pokretanje Kubernetes klastera u oblaku ovisno o poslovnim i drugim potrebama, arhitekturi sustava i sl.

Naravno, uvijek ostaje opcija pokretanja Kubernetes sustava na vlastitoj infrastrukturi u privatnom podatkovnom centru ili pak u nekoj od hibridnih verzija.

Kubernetes je moćan, ali vrlo složen sustav koji brzo mijenja način na koji organizacije izgrađuju, isporučuju i izvode moderne softverske sustave, proizvode, aplikacije i sl.

Njegove prednosti dolaze s povezanim sigurnosnim zahtjevima koji se moraju uzeti u obzir i riješiti zajedno s velikim brojem *open-source* (ali i komercijalnih) alata kako bi se smanjili rizici i prijetnje poslovanju.

Učinkovit pristup osiguravanju Kubernetes okruženja i aplikacija koje se u njemu izvode temelji se na primjeni sigurnosnih kontrola i rješenja za osiguranje sljedećih ključnih područja:

- Sigurnost lanaca alata koji se koristi za izradu slika spremnika, uključujući osnovne slike i komponente slika (eng. Images),
- Sigurnost infrastrukturnih komponente potrebne za pokretanje Kubernetes klastera, uključujući njegove ključne komponente i čvorove,
- Raspoređena i pokrenuta kontejnerska radna opterećenja sastavljena od pojedinačnih Pod-ova,
- Sigurnost mreže na svim razinama (Pod, čvor, klaster, …),
- Implementaciji nadzora, zapisivanja i revizije sustava Kubernetes.

Tehnologije spremnika nastavljaju širiti svoju ulogu u transformaciji načina na koji organizacije isporučuju svoje aplikacije. Uz sigurnost na razini Kubernetes klastera, dakle sa stanovišta sistemske administracije i arhitekture sustava Kubernetes koja je opisana u ovom specijalističkom radu, sve veća briga za sigurnošću sustava koji se baziraju na sustavu Kubernetes može se primijetiti i među DevOps timovima, odnosno u procesima nastanka aplikativnih rješenja, dakle tijekom *CI/CD* procesa.

Ono što je svakako dobro u tom smislu, a što se može i vidjeti u svim relevantnim istraživanjima, jest da DevOps timovi počinju implementirati sigurnost svojih aplikativnih rješenja tijekom samog procesa izgradnje aplikacije, dakle tijekom *CI/CD* procesa.

Međutim, potrebno je više rada na osiguranju samih spremnika kako bi se spriječilo da moguće ranjivosti uđu u sustav tj. da „probiju“ linije obrane.

Ključni trendovi naglašavaju kontinuirani rast spremničkih okruženja i sve veću ovisnost o rješenjima koja se temelje na otvorenom kodu. Ukratko, cijeli lanac izrade softvera od aplikacijskog koda do postavljenih spremnika kao i sigurnost Kubernetes klastera i svih njegovih komponenti mora biti osiguran.

Zbog složene strukture izvornih aplikacija u oblaku (eng. Cloud Native), svaki pojedini vektor napada mora se uzeti u obzir za sigurnu implementaciju aplikacija u sustav Kubernetes.

Implementacijom višeslojnog pristupa sigurnosti sustava, organizacije mogu s pouzdanjem skalirati svoju upotrebu sustava Kubernetes u produkcijskim okruženjima.

Stoga, iz svega dosad navedenoga možemo zaključiti da se sustav Kubernetes ubrzo nakon svoje pojave nametnuo kao standard za orkestraciju spremnika i igra ključnu ulogu u IT modernizaciji organizacija tj. poduzeća.

S obzirom na tu činjenicu sigurnost sustava Kubernetes nikako se ne smije zanemariti, dapače mora se uzeti u ozbiljno razmatranje i pokušati implementirati sve one sigurnosne zahtjeve, kontrole i sl. koje nameće arhitektura sustava, poslovni zahtjevi i ostali detalji sa ciljem što više razine sigurnosti cjelokupnog sustava Kubernetes u produkcijskom okruženju što je u većini implementacija sustava Kubernetes vrlo dugotrajan, složen i izazovan proces.

U zaključku ovog specijalističkog rada osvrnuti ćemo se na još jedno relevantno istraživanje iz studenoga 2022. godine koje je proveo DataDog (https://www.datadoghq.com/).

Istraživanje donosi analizu cjelokupnog „kontejner ekosistema“ i isto još jednom dokazuje dominaciju sustava Kubernetes u svijetu kontejnera, kontejner orkestratora i sl. na globalnoj razini. Istraživanje koje je proveo DataDog i dobiveni podaci mogu se vidjeti na slijedećem linku: https://www.datadoghq.com/container-report/.

11 ŽIVOTOPIS AUTORA

Tomislav Golubić, rođen je 1973. godine u Zagrebu. Osnovnu i srednju školu završava u Zagrebu. Diplomirao je na Šumarskom fakultetu, Sveučilišta u Zagrebu, 2000 godine. Oženjen je i otac jednog djeteta.

Od 2002. godine radi u Ministarstvu znanosti i tehnologije kao Viši informatički savjetnik u Službi za internu informacijsku infrastrukturu.

Nakon 15 godina rada u Ministarstvu znanosti i tehnologije, Ministarstvu znanosti, obrazovanja i športa, 2017. godine prelazi u Središnji državni ured za šport kao Savjetnik Državne tajnice za informacijske i komunikacijske tehnologije te kao Voditelj informatike u Središnjem državnom uredu za šport. U Središnjem državnom uredu za šport radi od 2017. – 2019. godine.

Od 2019. – 2020. godine radi u tvrtki 4Sec d.o.o. kao Sistem inženjer za mrežne i sigurnosne sustave.

Trenutno je zaposlen u Hrvatskoj lutriji d.o.o. u Sektoru za logistiku / Služba za informatiku kao Sistem inženjer za Linux i Windows operativne sustave.

Tijekom svoje karijere usavršavao se na mnogim informatičkim, informacijskim i komunikacijskim područjima i tehnologijama kao što su Red Hat Linux OS, Open Shift, MS Windows OS, Cisco, Elastic, Fortinet, Watchguard i sl.

Govori i služi se engleskim jezikom.

12 BIOGRAPHY OF THE AUTHOR

Tomislav Golubić was born in 1973 in Zagreb. He finished primary and secondary school in Zagreb. He graduated from the Faculty of Forestry, University of Zagreb, in 2000. He is married and the father of one child.

Since 2002, he has been working in the Ministry of Science and Technology as a Senior IT Advisor in the Service for Internal Information Infrastructure.

After 15 years of work in the Ministry of Science and Technology, the Ministry of Science, Education and Sports, in 2017 he moved to the Central State Office for Sports as Advisor to the State Secretary for Information and Communication Technologies and as Head of IT in the Central State Office for Sports. He worked in the Central State Office for Sports from 2017 to 2019.

From 2019 to 2020, he works in the company 4Sec d.o.o. as a System engineer for network and security systems.

He is currently employed at Hrvatska lutrija d.o.o. in the Logistics Sector / IT Department as a System Engineer for Linux and Windows operating systems.

During his career, he trained in many IT, information, communication and secitity fields and technologies such as Red Hat Linux OS, Open Shift, MS Windows OS, Cisco, Elastic, Fortinet, Watchguard, etc.

He speaks and uses the English language.

13 POPIS LITERATURE

[1.] B, Burns, J. Beda, K. Hightower „Kubernetes Up & Running“, OReilly, 2019.

[2.] L. Rice, M. Hausenblas „Kubernetes Security“, OReilly, 2019.

[3.] K. Huang „Learn Kubernetes Security“, Packt, 2020.

[4.] N, Kebbani, P. Tylenda, R. McKendrick, „The Kubernetes Bible“, Packt, 2021.

[5.] J. Rosso, R. Lander, A. Brand J, Harris, „Production Kubernetes“, OReilly, 2021.

[6.] A. Saleh, M. Karslioglu „Kubernetes in Production Best Practices“, Packt, 2021.

[7.] N. Poulton, „The Kubernetes Book“, Leanpub, 2022.

[8.] R. Chatterjee „Red Hat and IT Security“, Apress, 2021.

[9.] M. Hausenblas, „Container Networking“, OReilly, 2018.

[10.] D. A. Tevault „Mastering Linux Security and Hardening“, Packt, 2020.

[11.] M. Pérez Colino, P. Gómez Scott McCarty „Red Hat Enterprise Linux 8 Administration“, Packt, 2021.

14 POPIS INTERNETSKIH IZVORA

[1.] Kubernetes, dostupno na: https://kubernetes.io/ (23.12.2022.)

[2.] Minikube, dostupno na: https://minikube.sigs.k8s.io/docs/ (11.10.2022.)

[3.] Red Hat, dostupno na: https://www.redhat.com/ (zadnji puta posjećeno: 17.12.2022.)

[4.] OpenSource, dostupno na: https://opensource.com/ (17.12.2022.)

[5.] Linux Fondation, dostupno na: https://linuxfoundation.org/ (21.12.2022.)

[6.] Docker, dostupno na: https://www.docker.com/ (7.11.2022.)

[7.] Threat model, dostupno na: https://en.wikipedia.org/wiki/Threat_model (21.12.2022.)

[8.] MITRE, dostupno na: https://cve.mitre.org/ (20.12.2022.)

[9.] Microservices, dostupno na: https://microservices.io/ (20.12.2022.)

[10.] Udemy, dostupno na: https://www.udemy.com/ (20.12.2022.)

[11.] Packt, dostupno na: https://www.packtpub.com/ (23.12.2022.)

[12.] OReilly, dostupno na: https://www.oreilly.com/ (20.12.2022.)

[13.] CNCF, dostupno na: https://www.cncf.io/ (21.12.2022.)

[14.] Kube Academy, dostupno na: https://kube.academy/ (21.12.2022.)

[15.] YAML, dostupno na: https://yaml.org/ (21.12.2022.)

[16.] Elastic, dostupno na: https://www.elastic.co/elastic-cloud-kubernetes (21.12.2022.)

[17.] Prometheus, dostupno na: https://prometheus.io/ (21.12.2022.)

[18.] Grafana, dostupno na: https://grafana.com/ (21.12.2022.)

[19.] Kibana, dostupno na: https://www.elastic.co/kibana/ (21.12.2022.)

[20.] Lens IDE, dostupno na: https://k8slens.dev/ (21.12.2022.)

[21.] CIS, dostupno na: https://www.cisecurity.org/benchmark/kubernetes (21.12.2022.)

[22.] Kube Bench, dostupno na: https://github.com/aquasecurity/kube-bench (21.12.2022.)

[23.] Kubescape, dostupno na: https://github.com/kubescape/kubescape (23.12.2022.)

[24.] Kubernetes auditing, dostupno na: https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/ (21.12.2022.)

[25.] Kubernetes Hardening Guide, dostupno na: https://media.defense.gov/2022/Aug/29/2003066362/-1/-1/0/CTR_KUBERNETES_HARDENING_GUIDANCE_1.2_20220829.PDF (17.12.2022.)