# Kubernetes v omrežjih z omejenim dostopom

Benjamin Burgar, Uroš Brovč, Urban Zaletel

Kontron d.o.o, Kranj, Slovenija
benjamin.burgar@kontron.si, uros.brovc@kontron.si, urban.zaletel@kontron.si

Namestitev aplikacij na Kubernetes je preprosta, kadar imamo dostop do javnih ali privatnih registrov. Vendar se pri upravljanju Kubernetes gruče v omrežjih z omejenim dostopom pojavljajo izzivi. Kubernetes uporablja zabojnike, Helm chart-e in druge artefakte, ki jih med delovanjem pridobiva iz internetnih in privatnih registrov. Te artefakte moramo zagotoviti tudi v omrežjih z omejenim dostopom. Pri vzdrževanju več identičnih Kubernetes gruč v takšnih omrežjih se soočamo s problemom sledenja verzij aplikacij in njihove konfiguracije. V članku je predstavljen način namestitve aplikacij na Kubernetes gruče v omrežjih z omejenim dostopom. Prikazana je rešitve za prenos zabojnikov in Helm chart-ov v takšna okolja ter kako lahko brez centralnega sistema za verzioniranje zagotovimo deklarativno namestitev aplikacij. Poudarek je tudi na varnostnem pregledu vseh komponent rešitve in natančnemu sledenju in upravljanju vseh komponent našega sistema, kar je ključno za zanesljivo in ponovljivo namestitev rešitve na različna okolja.

**Ključne besede:**
Kubernetes
privatna omrežja
DevOps
Helm
zabojnik
registri programske opreme
5G

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.14

# 1 Uvod

V 5G privatnih omrežjih pogosto ni omogočenega dostopa do interneta zaradi varnostnega vidika podjetij, kjer se ta omrežja nameščajo. To pomeni, da ni mogoče uporabljati javnih registrov programske opreme, kar predstavlja velik izziv pri nameščanju in upravljanju aplikacij ter komponent, ki se zanašajo na te vire.

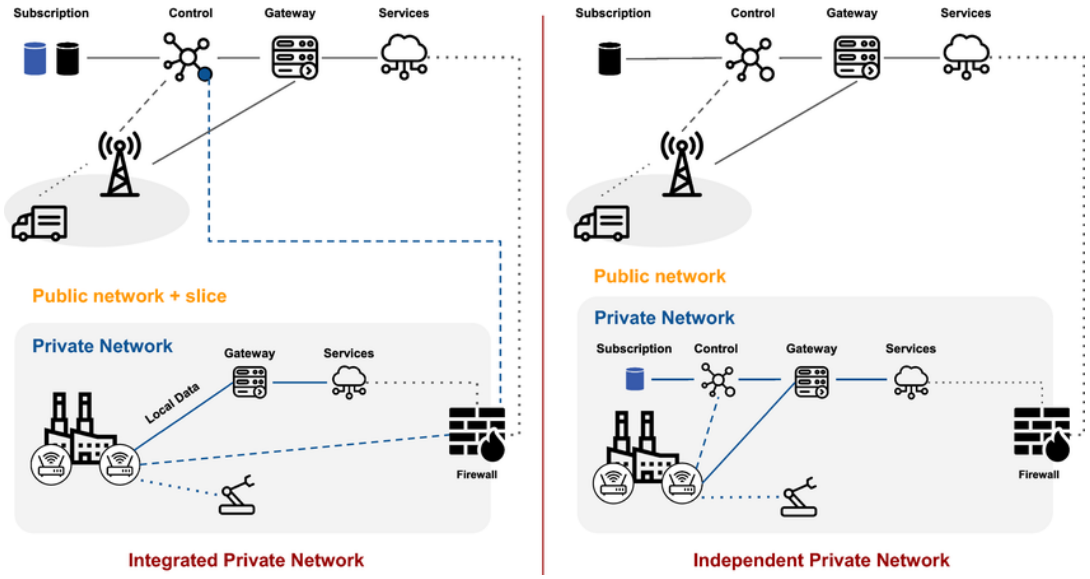
Da bi rešili ta problem, smo razvili rešitev, ki omogoča deklarativno generacijo programskih paketov, njihov prenos in namestitev pri stranki. Dodatno varnost zagotavljamo z rednimi varnostnimi pregledi vseh komponent rešitve pred generacijo paketa. Ta pristop zagotavlja, da so vse potrebne komponente na voljo tudi v primeru omejenega dostopa do interneta, s čimer se poenostavi upravljanje in vzdrževanje 5G privatnih omrežij.

# 2 5G privatna omrežja

## 2.1. Namen

5G privatna omrežja v podjetjih se uporabljajo zaradi več razlogov. Omogočajo hitrejše in bolj zanesljive komunikacijske povezave, kar je ključno za podjetja, ki potrebujejo visoko razpoložljivost in nizko latenco za svoje aplikacije in storitve. Privatna omrežja podjetjem nudijo večji nadzor in varnost nad njihovimi podatki ter omrežnimi operacijami. Poleg tega omogočajo prilagodljivost in prilagojene rešitve, ki ustrezajo specifičnim potrebam podjetja, kot so pametne tovarne, IoT naprave in avtomatizacija procesov.

5G privatna omrežja so lahko integrirana z javnim 5G omrežjem. Npr. Operater postavi javno omrežje in preko segmentacije omogoči ločevanje kontrolnega in uporabniškega prometa. Druga možnost je postavitev neodvisnega 5G privatnega omrežja, v določenih primerih z omejenim dostopom.




**Slika 1: Primerjava integriranega in neodvisnega 5G privatnega omrežja.** [1]

### 2.2. Omrežja z omejenim dostopom

Za podjetja je ključnega pomena, da z uvajanjem 5G privatnega omrežja ne ogrozijo varnostne politike. Pogosta zahteva je, da se v 5G omrežju, kjer se nameša rešitev, ne omogoča internetnega dostopa. To zagotavlja zaščito pred zunanjimi grožnjami, kot so hekerski napadi in nepooblaščeni dostopi, kar je še posebej kritično za podatkovno občutljiva podjetja. Še posebno občutljiva so OT omrežja, ta omrežja so največkrat povsem ločena od preostale IT infrastrukture. Onemogočanje internetnega dostopa v 5G privatnih omrežjih pomaga ohranjati integriteto podatkov in omrežne infrastrukture ter zagotavlja, da so podatki podjetja ustrezno zaščiteni in varni.

## 3 Namestitev rešitve v omrežju z omejenim dostopom

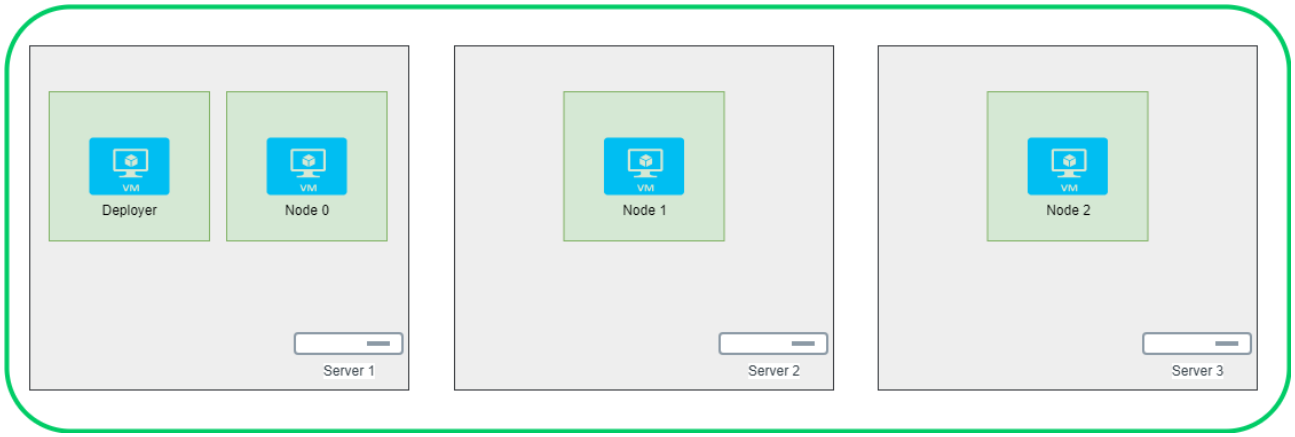
Če na lokaciji, kjer želimo namestiti rešitev, ni dostopa do interneta, je potrebno zagotoviti prenos vseh artefaktov in namestitev lokalno. V primeru, ko izvajamo celotno namestitev od inštalacije strežnikov do končne rešitve je potrebno zagotoviti zbiranje in pakiranje tudi za ostale komponente rešitve (operacijski sistem, Kubernetes[8], itd). V nadaljevanju se bomo bolj podrobno posvetili nameščanju aplikacije na Kubernetes gručo in specifiki pri omejenem dostopu.

Za zagotavljanje lokalne inštalacije je pomembno, da že med pripravo rešitve oziroma paketa zberemo vse potrebne komponente, kot so različne skripte, slike zabojnikov, Helm chart-i[9], itd. Več o tem v poglavju 4.

### 3.1. Okolje in način namestitve

Opis inštalacije je podan za okolje, kjer imamo že postavljene strežnike z ustreznim operacijskim sistemom. Podan je primer za HA[10] postavitev naše 5G rešitve.

Za namen namestitve in upravljanja rešitve se na enem izmed strežnikov kreira navidezna naprava za nameščanje rešitve (Deployer). Za namestitev Kubernetes gruče se na vsakem strežniku kreira navidezno napravo (Node 0, Node 1 in Node 2).

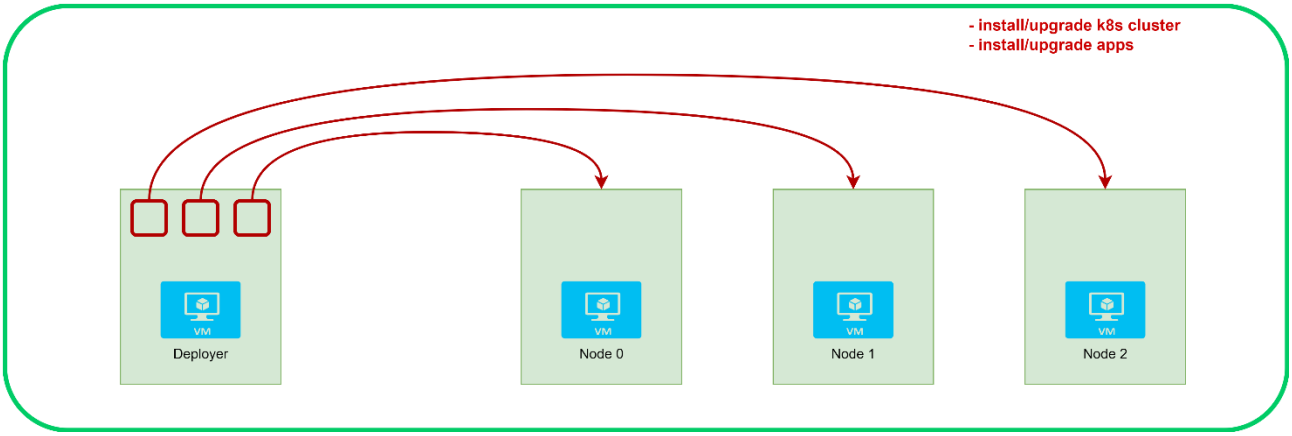



**Slika 2: Virtualne naprave.**

Namestitev Kubernetes gruče se izvede preko navidezne naprave Deployer. Za ta namen so ustvarjene avtomatizacijske skripte, ki zagotavljajo kontroliran postopek namestitve na različnih objektih oziroma lokacijah.

[8] https://kubernetes.io/
[9] https://helm.sh/docs/topics/charts/
[10] HA – izvorno »High Availability«, Visoka razpoložljivost

Po uspešno nameščeni in konfigurirani Kubernetes gruči se po točno določenem postopku izvede namestitev aplikacij. Tudi v tem primeru se namestitev izvaja preko navidezne naprave Deployer. Avtomatizacijske skripte poskrbijo za enostaven in ponovljiv proces namestitve, kar zmanjšuje možnosti napak in zagotavlja doslednost.




**Slika 3: Namestitev k8s in aplikacij.**

Pri omenjenem postopku namestitve je predvideno, da imamo dostop do interneta oziroma do javnih in zasebnih registrov. Vse artefakte se pridobi iz omenjenih registrov med izvedbo inštalacije. V primeru omejenega dostopa to ni omogočeno, zato je bilo potrebno ustrezno dopolniti oziroma spremeniti postopek inštalacije. Dopolnitev postopka je opisana v poglavju 3.2.

Ključno je tudi, da se na lokaciji namestitve omogoči pregled verzije nameščenih komponent in uporabljene konfiguracije. To nam omogoča pregled podprtih oz. omogočenih funkcionalnosti, pomaga pri reševanju napak s strani razvoja (na kateri verziji programske opreme se je napaka pojavila, kakšna je konfiguracija), pri posodabljanju verzij, itd. Več o tem v poglavju 3.5.

## 3.2. Lokalna namestitev brez internetnega dostopa

V primeru lokalne namestitve brez dostopa do interneta je potrebno prenesti vse zahtevane pakete preko medija (npr. USB ključka, prenosnika) in jih shraniti na navidezno napravo za namestitev rešitve (Deployer). Seznam paketov in sami paketi so pripravljeni med pripravo programske opreme za posamezno verzijo rešitve, kot je opisano v poglavju 4. Paketi so v našem primeru pripravljeni na centralnem registru artefaktov (Nexus[11]).

Za varen prenos paketov in tar datotek preko USB medija je pomembno zagotoviti, da datoteke ostanejo nespremenjene. To dosežemo z uporabo digitalnih podpisov, ki nam omogočajo, da preverimo, ali so datoteke ostale nedotaknjene in niso bile spremenjene ali kompromitirane med prenosom.

Ko so na voljo oz. prenešeni vsi paketi, je potrebno namestiti lokalni OCI[12] register, v kolikor še ni nameščen, na navidezno napravo Deployer. Namestitev se izvede s skripto, ki je vključena v prej omenjenih paketih. Glej poglavje 3.3 za dodaten opis OCI registra.

V nameščen lokalni OCI register se prenese artefakte iz prenesenega paketa (tar datoteka) na navidezni napravi Deployer. Za ustrezen uvoz paketa (oz. vseh artefaktov v paketu) v register se uporabi lastno orodje (regcli), ki nam poenostavi oz. omogoči različne operacije nad lokalnih registrom. Združevanje in kopiranje artefaktov med različnimi registri ali datotekami je opisano v poglavju 3.4.

[11] https://www.sonatype.com/products/sonatype-nexus-repository
[12] OCI – Open Container Initiative

Namestitveni postopek za Kubernetes gručo in kasneje za aplikacije je od te točke dalje podoben kot v primeru dostopa do interneta. Ključna razlika je, da se vsi zahtevani artefakti iščejo v lokalnem registru in ne več v javnem ali privatnem registru preko internetne povezave.

V kolikor se za Kubernetes runtime uporablja containerd[13], ni potrebno izvajati dodatnih sprememb na aplikaciji oz. v Helm chart-ih. Namreč konfiguracija containerd za registre omogoča preslikavo naslovov. Praktično to pomeni, da se v konfiguraciji zapiše na kakšen naslov se preslikajo določeni naslovi za javne registre. Na primer, naslednja konfiguracija bi potegnila sliko iz lokalnega registra (primer naslova: https://registry.example.com:5000) v primeru uporabe imena library/busybox, kot tudi registry.example.com/library/busybox:latest:

```
mirrors:
  docker.io:
    endpoint:
      - "https://registry.example.com:5000"
  registry.example.com:
    endpoint:
      - "https://registry.example.com:5000"
```

**Slika 4: Containerd konfiguracija.**

V kolikor se uporablja docker runtime, je potrebno to upoštevati tudi pri aplikacijah in omogočiti spremembo naslovov za registre v samem Helm chart-u med inštalacijo (možnost konfiguracije v values.yaml).

### 3.3. OCI register

OCI (Open Container Initiative) register se uporablja za standardizirano shranjevanje, distribucijo in upravljanje slik zabojnikov in drugih artefaktov, kot so Helm chart-i. Zagotavlja interoperabilnost med različnimi orodji in platformami, omogoča varno shranjevanje in prenos podatkov ter podpira avtomatizirane procese neprekinjene integracije in dostave (CI/CD).

Obstaja več odprtokodnih izvedb registrov, ki ustrezajo OCI specifikaciji, kot na primer: Distribution, Harbor, Zot, itd.

V našem primeru smo se odločili za Distribution register[14]. V register se shranjuje slike zabojnikov in Helm chart-e. Lokalni register Distribution teče v zabojniku na navidezni napravi Deployer. Za pregled, dodajanje ali brisanje artefaktov iz registra se uporablja lastno razvito orodje (regcli), ki komunicira z registrom preko HTTP API-ja (posredno tudi preko skopeo orodja).

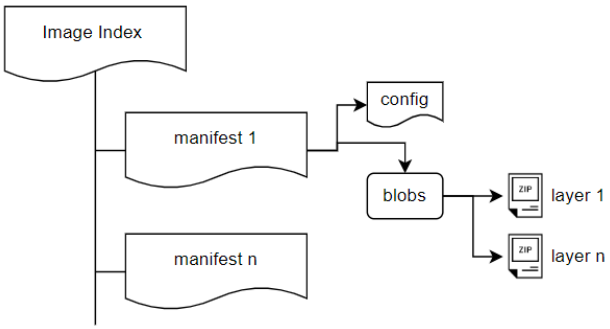
V registru so vsi artefakti strukturirani, kot je določeno v specifikaciji za OCI slike[15]. S tem se zagotavlja standardiziran format za deljenje, distribucijo in izvajanje aplikacij preko različnih platform in orodij.

Specifikacija slike Open Container Initiative (OCI) določa format in strukturo slik za zabojnike. Glavni deli specifikacije OCI slike vključujejo:

- **Postavitev slike (Image Layout)**: Opisuje strukturo datotečnega sistema, ki vsebuje OCI sliko. To vključuje organizacijo slike, datoteke in imenike, ki jih vsebuje, ter pravila za shranjevanje vsebine in metapodatkov slike.
- **Manifest slike (Image Manifest)**: JSON datoteka, ki opisuje komponente slike, kot so sloji, konfiguracija in specifične podatke za platformo. Vključuje:
    - **Config**: Sklic na konfiguracijski objekt, ki opisuje nastavitve delovanja kontejnerja.
    - **Layers**: Urejen seznam sklicev na sloje slike, ki vsebujejo dejansko vsebino datotečnega sistema.

[13] https://containerd.io/
[14] https://github.com/distribution/distribution/
[15] https://github.com/opencontainers/image-spec

- **Kazalo slike (Image Index)**: Izbirna JSON datoteka, ki podpira slike za več platform. Vsebuje sklice na več manifestov slik, kar omogoča podporo za različne arhitekture ali operacijske sisteme.
- **Konfiguracija slike (Image Configuration)**: JSON objekt, ki določa konfiguracijo delovanja kontejnerja, vključno s podrobnostmi, kot so vstopna točka, okoliške spremenljivke, delovni imenik in omejitve virov.
  - **Sloj (Layer)**: Predstavlja del datotečnega sistema zabojnika. Sloji so zloženi en na drugega, da ustvarijo celoten datotečni sistem za zabojnik. Vsak sloj je stisnjeni tar arhiv.
  - **Deskriptorji (Descriptors)**: Objekti, ki opisujejo vsebino (kot so sloji, konfiguracije itd.) znotraj slike, vključno z lastnostmi, kot so medijski tip, velikost in digest[16].
  - **Opombe (Annotations)**: Ključ-vrednost pari, ki zagotavljajo dodatne metapodatke za slike, manifeste in konfiguracije. Opombe lahko vključujejo informacije kot so avtor, opis in različica.
  - **Medijski tipi (Media Types)**: Specifikacije za vrste vsebine, ki so lahko vključene v OCI sliko, kot so medijski tipi slojev, konfiguracij in manifestov. To zagotavlja skladnost pri interpretaciji podatkov.

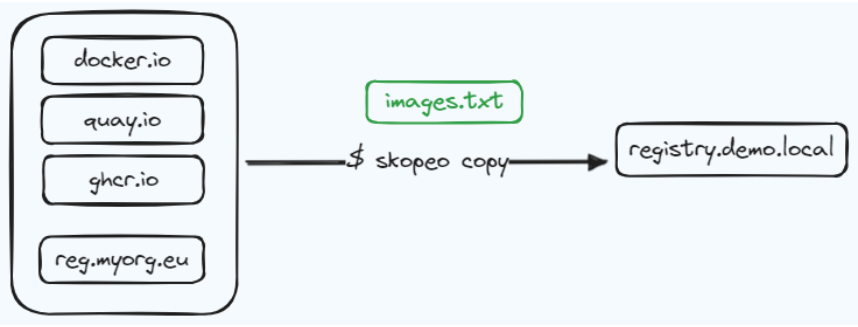



**Slika 5: Struktura OCI slike.**

### 3.4. Kopiranje artefaktov med različnimi registri

Prenos paketov oz. artefaktov na določeno oddaljeno lokacijo oz. v lokalen register lahko izvedemo na dva načina.

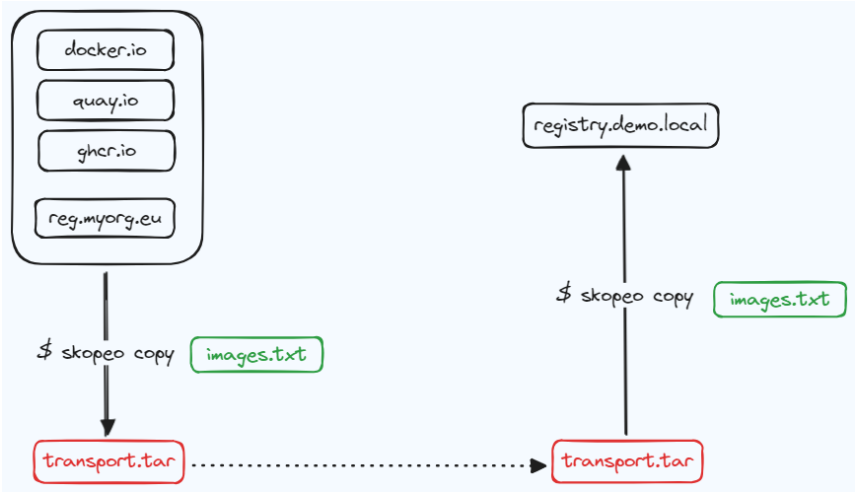
Preko varne (VPN) povezave iz privatnega in ostalih javnih registrov v lokalen register na lokaciji, kjer se bo nameščalo rešitev.




**Slika 6: Prenos slik zabojnikov iz različnih registrov v lokalen OCI register.**

Pakete se lahko prenese tudi z različnimi mediji oz. napravami (USB ključek, prenosnik). Pred prenosom se izvede pakiranje artefaktov, nato prenos in na koncu odpiranje in vključevanje artefaktov v lokalen register. Omenjen način je uporabljen v primeru omejenega dostopa do interneta.

[16] digest - kriptografski hash, ki zagotavlja celovitost vsebine

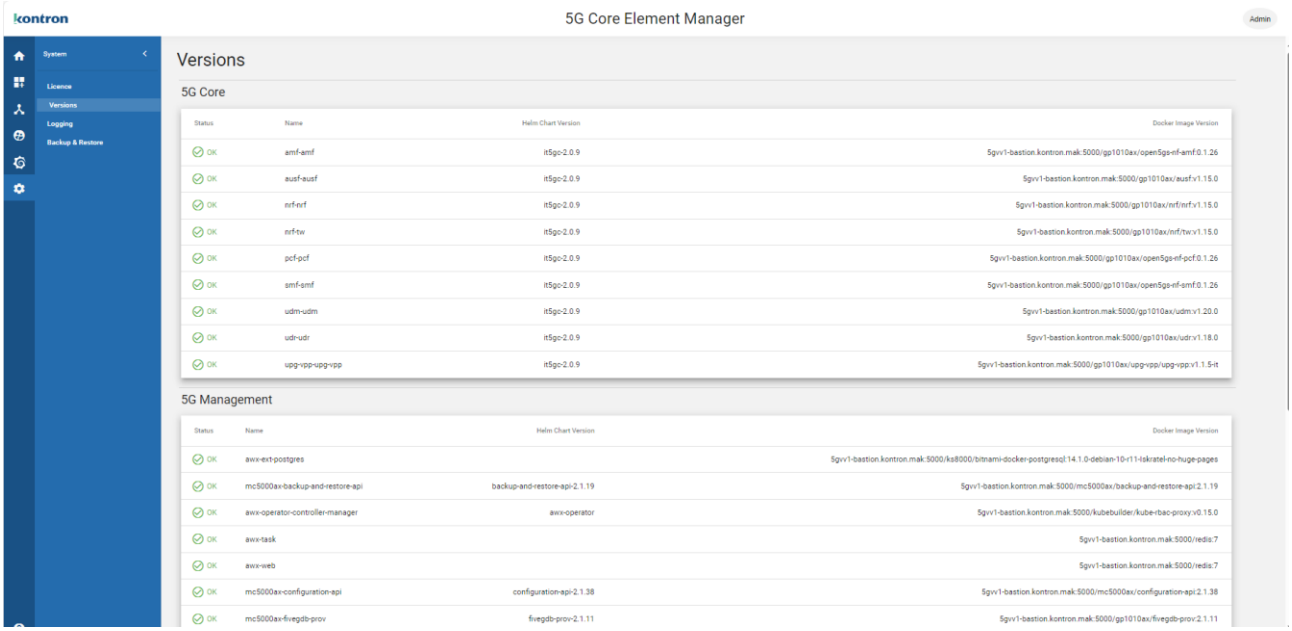



**Slika 7: Pakiranje, prenos in odpiranje paketa v lokalen OCI register.**

Za prenos OCI artefaktov obstajajo razna odprtokodna orodja. Za prenos docker slik lahko uporabljamo skopeo, za prenos Helm chart-ov lahko uporabljamo helm. Obstajajo tudi orodja, ki poenotijo prenos obeh vrst artefaktov, na primer projekta Crane[17] in Mindthegap[18].

### 3.5. Izpis uporabljenih komponent rešitve

Skupek verzij nameščenih aplikacij je vodeno preko verzije rešitve. Posamezna verzija rešitve določa točno katere komponente so del te rešitve. Za hitro preverjanje verzije 5G rešitve oz. uporabljenih aplikacij in komponent smo na rešitvi omogočili priročen pregled preko uporabniškega vmesnika.



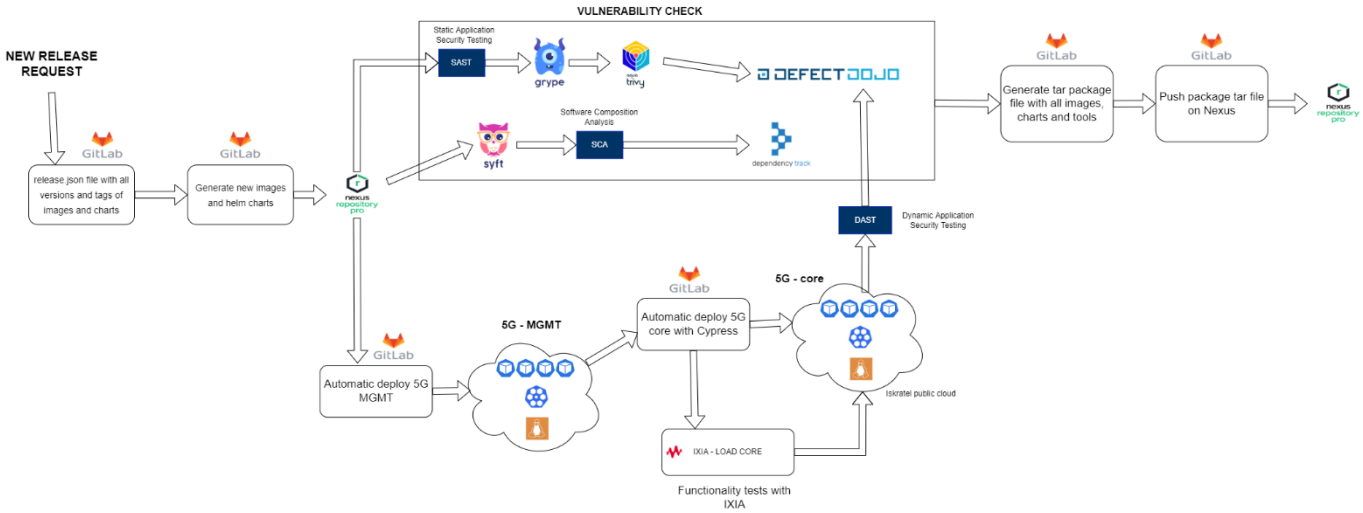

**Slika 8: Pregled verzij preko uporabniškega vmesnika.**

Verzijo rešitve ter posameznih komponent se lahko preveri tudi preko navidezne naprave Deployer z uporabo helm ukazov (helm list, helm show values).

[17] https://github.com/google/go-containerregistry/tree/main/cmd/crane
[18] https://github.com/mesosphere/mindthegap

# 4 Priprava programske opreme

Za zagotovitev sledljivosti rešitve in vseh vključenih komponent ter zanesljive generacije novih paketov, smo omogočili avtomatizirano generacijo in objavo verzij rešitve. Z ustreznimi CI/CD postopki oz. cevovodom se izvede preverjanje, generacija in obveščanje uporabnikov o novi verziji rešitve. Zahtevek za novo verzijo rešitve se proži ročno. Pred zagonom cevovoda je potrebno določiti verzije vseh komponent ter verzijo rešitve.




**Slika 9: Cevovod za novo verzijo rešitve.**

V primeru uspešno izvedenega cevovoda, pridobimo pakete za novo verzijo rešitve ter poročilo s seznamom vseh vključenih komponent, povezavami do vseh kreiranih paketov in skript, seznam rešenih oz. dodanih funkcionalnosti na rešitvi, inštalacijska navodila, uporabniška navodila, itd.

V cevovodu se preverja tudi varnostne ranljivosti. V primeru ugotovljenih večjih ranljivosti se smatra cevovod kot neuspešen kar pomeni, da verzija rešitve ni ustrezna za objavo. Več o varnostnem preverjanju v poglavju 4.1.

## 4.1. Varnostno preverjanje

GitLab CI/CD, Grype, Trivy, Syft, DefectDojo, Dependency Track, IXIA in UERANSIM so ključna orodja, ki nam omogočajo zagotavljanje najvišje ravni varnosti in kakovosti kode. Osrednje načelo DevSecOps[19] je integracija varnostnih preverjanj v celoten CI/CD[20] proces, kar zagotavlja, da varnost ni zgolj dodatek, temveč bistven del razvoja.

V cevovodu se izvaja statično analizo kode, izdelavo SBOM in teste ranljivosti na slikah zabojnikov, kar nam omogoča hitro odkrivanje in odpravljanje najnovejših varnostnih pomanjkljivosti, kot jih določa CVE[21]. Za zagotavljanje celovite varnosti se nenehno spremlja, dokumentira in odpravlja vse odkrite ranljivosti.

Na začetku cevovoda izvedemo popis programske opreme SBOM[22], kar nam praktično predstavlja inventar programske opreme. Nujna je pridobitev polne oblike SBOM, kar omogoča, da ranljivosti ugotovimo takoj. Obstaja več standardnih formatov za izmenjavo SBOM, kar omogoča predvsem interoperabilnost med različnimi orodji. V našem primeru uporabljamo format CycloneDX. Z izdelavo SBOM si zagotovimo natančno sledenje in upravljanje vseh komponent našega sistema.

[19] DevSecOps – Development, Security and Operations
[20] CI/CD – Continuous Integration/Continuous Delivery
[21] https://cve.mitre.org/
[22] SBOM – Software Bill Of Materials

Za statično analizo Golang programske kode uporabljamo GOLANGCI-LINT, za analizo kode drugih programskih jezikov (kot so Java in C) pa uporabljamo integrirano orodje v GitLab-u[23]. Statično analizo kode se izvaja ob vsakem potisku kode na GitLab repozitorij.

Skeniranje ranljivosti in pregledovanje slik vsebnikov izvajamo z orodjem GRYPE katero pridobi vse potrebne varnostne podatke iz svoje zaledne oblačne storitve. Slika vsebnika lahko vsebuje hrošče in varnostne ranljivosti, ki se lahko skozi proces gradnje dolgoročno ponavljajo in s tem povečajo tveganje za vse različice pakirane programske opreme. Posledično je ključnega pomena izvajanje rednega pregleda slik vsebnikov in zgodnja odprava težav. Poleg orodja GRYPE za pregledovanje slik in vsebnikov izvajamo pregledovanje tudi z orodjem Aqua Trivy, ki je zelo podobno orodju GRYPE. Glavna razlika med orodji je, da slednje uporablja lastno kompaktno bazo, ki se lahko redno posodablja.

Za upravljanje z varnostnimi ranljivostmi uporabljamo Defect Dojo, kateri je kot centralna baza ranljivosti. Poenostavi postopek njihovega spremljanja skozi proces razvoja, delegiranja odgovornim osebam in njihove odprave oz. zapiranje posamezne ranljivosti. Omogoča uvoz varnostnih poročil iz zunanjih orodij in sistemov, združevanje in odstranjevanje dvojnikov ter ustvarjanje poročil. DefectDojo nadgradi postopek upravljanja ranljivosti prek več internih modelov, ki jih je mogoče manipulirati s programsko kodo. Osnovni modeli vključujejo: »engagements«, »tests« in »findings«, na voljo pa je še več modelov, ki olajšajo spremljanje metrik, avtentikacijo in ustvarjanje poročil.

OWASP DependencyTrack je platforma za analizo komponent, ki organizacijam omogoča prepoznavanje in zmanjšanje tveganja v dobavni verigi programske opreme. DependencyTrack uporablja pristop sledenja odvisnosti na osnovi SBOM. Orodje samo po sebi ne ustvarja SBOM-ov, ampak predstavlja platformo oz. podatkovno bazo, ki omogoča zbiranje in analizo SBOM-ov, iskanje po gradnikih in verzijah, sledenje odprtokodnih licenc, ipd. S tem omogoča hitro ukrepanje ob varnostnih incidentih.

## 5 Zaključek

V prispevku smo opisali postopek namestitve 5G jedrnega omrežja v okolju brez dostopa do interneta. Ključni poudarek je na pomenu sledenja uporabljenim komponentam, varnostnem preverjanju artefaktov, varnem prenosu artefaktov na končno lokacijo in zagotavljanju deterministične namestitve. Varni prenos artefaktov zagotavlja, da so vse nameščene komponente preverjene in da ni prišlo do kompromitacije med transportom. Poleg tega potrebno imeti dober pregled nad nameščenimi verzijami komponent in konfiguracijami, saj to omogoča učinkovito upravljanje in vzdrževanje sistema, ter hitrejšo identifikacijo težav oziroma potrebnih posodobitev na posamezni postavitvi. Z upoštevanjem teh praks lahko zagotovimo stabilno in varno delovanje 5G jedrnega omrežja, tudi v okoljih, kjer dostop do interneta ni mogoč. Zanesljivost in robustnost omrežja je ključno za podporo sodobnim aplikacijam in storitvam, ki temeljijo na 5G tehnologiji.

## Literatura

[1] https://www.researchgate.net/figure/Integrated-vs-Independent-Private-5G-Networks_fig5_365608799, Integrated vs Independent Private 5G, obiskano 8.7.2024

[2] https://docs.rke2.io/install/containerd_registry_configuration, Containerd Registry Configuration, obiskano 10.7.2024

[3] https://ravichaganti.com/blog/2022-10-28-understanding-container-images-oci-image-specification/, Understanding container images - OCI image specification, obiskano 11.7.2024

[23] Gitlab static application security testing - https://docs.gitlab.com/ee/user/application_security/sast/