PROJEKT

Multitenantné operačné centrum kybernetickej bezpečnosti riešené
ako otvorená cloudová služba s prvkami strojového učenia

Previazané s balíkom KPB4 – Architektúra riešenia

Previazané s výstupom
V10 – Zoznam funkčných a nefunkčných požiadaviek
a V11 – Biznis architektúra

Financované
Európskou úniou
NextGenerationEU

PLÁN OBNOVY

ÚRAD VLÁDY
SLOVENSKEJ REPUBLIKY

Obsah

1. POROZUMENIE KONTAJNERIZÁCII

1.1. DEFINÍCIA KONTAJNERIZÁCIE

Kontajnerizácia je virtualizačná technológia, ktorá umožňuje spúšťať a nasadzovať aplikácie a služby bez potreby vytvárania samostatného virtuálneho servera pre každé riešenie. Využíva tzv. kontajnery — izolované, ľahké, rýchlo spúšťané prostredia, ktoré zdieľajú jadro operačného systému, ale obsahujú len tie zdroje, ktoré aplikácia skutočne potrebuje.

Historický vývoj

- Koncept kontajnerov sa objavil už v 70. rokoch v Unix systémoch na izoláciu procesov (napr. chroot).
- Väčšie rozšírenie priniesli Linux kontajnery v 2000-tych rokoch.
- Nasledovali nástroje pre orchestráciu ako Apache Mesos, Google Borg či Facebook Tupperware, ktoré umožnili spravovať stovky kontajnerov.
- Masový boom nastal po predstavení Dockeru v roku 2013, čo zjednodušilo použitie kontajnerov a rozšírilo ich v komerčnom prostredí.
- Na Docker nadviazali platformy Kubernetes, Marathon či Docker Swarm, ktoré riešia automatizáciu, správu a orchestráciu kontajnerov vo veľkom rozsahu.

1.2. OPERAČNÝ SYSTÉM A RUNTIME

Operačný systém (OS) poskytuje programy, knižnice, nástroje a prostredie, ktoré riadia chod počítača a umožňujú spúšťať aplikácie.

- Runtime prostredie zabezpečuje aktívne vykonávanie aplikácií.
- Samotná aplikácia si môže prinášať aj vlastný runtime, ktorý nadväzuje na ten operačný.

1.3. ZÁKLADY VIRTUALIZÁCIE

Virtualizácia umožňuje rozdeliť fyzický server na viacero virtuálnych serverov:

- **Virtuálny server** je plnohodnotná kópia operačného systému, bežiaca na virtualizovanom hardvéri.
- Typy hypervisorov:
  - **Typ 1** (bare metal): hypervisor beží priamo na hardvéri bez OS
  - **Typ 2**: hypervisor beží na už existujúcom OS
- Hypervisor vytvára virtuálne servery a riadi ich činnosť.

1.4. ROZDIEL MEDZI VIRTUALIZÁCIOU A KONTAJNERIZÁCIOU

- Virtuálny server simuluje celý operačný systém.
- Kontajner využíva len potrebné komponenty OS a zdieľa jadro s hostiteľským OS, čo je efektívnejšie.
- Kontajnery majú nižšiu spotrebu pamäte, rýchlejšie sa spúšťajú a sú prenosnejšie.

Kontajnerizácia na fyzických serveroch
Kontajnerizačná platforma (napr. Docker Engine) beží priamo na hostiteľskom OS. Kontajnery využívajú subset OS prostriedkov a sú izolované, ale nie je potrebná ďalšia vrstva virtualizácie.

Kontajnerizácia na virtuálnych serveroch
Kontajnery môžeme prevádzkovať aj vo virtuálnych serveroch, čo kombinuje výhody virtualizácie (bezpečnostná segmentácia) s flexibilitou kontajnerov.

- Typ 1 virtualizácia je preferovaná v produkčnom prostredí
- Typ 2 virtualizácia je bežná v testovaní a vývoji

Business a technologické motivácie

- Potreba rýchlejšie reagovať na zmeny trhu (agilita)
- Znižovanie nákladov na infraštruktúru
- Lepšie zvládanie špičkovej prevádzky
- Cloud computing a Infrastructure-as-Code zrýchlili prijatie kontajnerizácie
- Štandardizované open-source prostredia

Výhody kontajnerizácie

- Efektívnejšie využívanie zdrojov
- Lepšia prenositeľnosť (portable)
- Optimalizácia riešení
- Jednoduchšia škálovateľnosť
- Vyššia odolnosť (resiliency)
- Rýchlejšie nasadenie
- Podpora automatizácie (CI/CD)

Riziká a výzvy

- Bezpečnostné otázky (napr. zdieľané jadro)
- Potreba zabezpečiť správne nastavenie siete
- Sledovanie verzií a konfigurácií
- Monitoring kontajnerov v prevádzke

2. ZÁKLADNÉ POJMY A KONCEPTY

Táto lekcia podrobne vysvetľuje základnú terminológiu a pojmy, ktoré sú kľúčové pri práci s kontajnermi a kontajnerizačnými platformami.

2.1. KĽÚČOVÉ POJMY

- **Kontajner**
  Virtuálne hosťovacie prostredie optimalizované tak, aby poskytlo len zdroje, ktoré konkrétny softvér potrebuje. Kontajner je izolovaný od iných procesov, ale zdieľa hostiteľské jadro operačného systému.
- **Kontajnerový obraz (container image)**
  Je to vopred definovaná šablóna (template), na základe ktorej sa kontajnery vytvárajú. Ukladá sa v repozitári a predstavuje všetky vrstvy potrebné na vytvorenie kontajnera.
- **Kontajnerový engine (container engine)**
  Softvérová platforma, ktorá vytvára kontajnery na základe kontajnerových obrazov, riadi ich beh a spravuje ich životný cyklus. Príkladom je Docker Engine.
  - engine má **management plane** (rozhranie pre administrátorov)
  - a **control plane** (automatické riadiace mechanizmy)
- **Pod (logical pod container)**
  Špeciálny typ kontajnera, ktorý môže obsahovať jeden alebo viac kontajnerov, ktoré zdieľajú konfiguráciu, úložisko a sieťové prostriedky. Pod sa spravuje ako jeden celok a používa sa v platformách ako Kubernetes.
- **Hostiteľ (host)**
  Server (fyzický alebo virtuálny), na ktorom sú kontajnery spustené. Hostiteľ poskytuje operačný systém a prostriedky pre kontajnery.
- **Hostiteľské klastry**
  Zoskupenie viacerých hostiteľov (serverov), ktoré poskytujú kolektívnu výpočtovú kapacitu. Môže ísť o:
  - **Load balanced cluster** (vyvažovanie záťaže)
  - **High availability cluster** (vysoká dostupnosť)
  - **Scaling cluster** (škálovanie)
- **Hostiteľské siete a overlay siete**
  Kontajnery na jednom hostiteľovi komunikujú cez lokálnu hostiteľskú sieť, zatiaľ čo kontajnery na rôznych hostiteľoch komunikujú cez overlay siete. Tieto siete zabezpečujú konektivitu medzi kontajnermi v rámci jednej aplikácie alebo riešenia.

2.2. DETAILNEJŠIE VYSVETLENIE

- **Kontajnery** sú veľmi flexibilné, pretože sa spúšťajú rýchlejšie než klasické virtuálne servery, ich image je menší a dajú sa efektívne presúvať medzi prostrediami (napr. test, produkcia).

- **Kontajnerový obraz** je nemenný (immutable). Ak je potrebná zmena, vytvorí sa nový obraz.
- **Pod** umožňuje viacerým kontajnerom, ktoré spolu úzko súvisia, zdieľať IP adresu a úložisko.
- **Hostiteľské klastry** umožňujú vyššiu dostupnosť a škálovateľnosť, pretože ak vypadne jeden server, jeho úlohy môže okamžite prevziať iný v klastri.

Prepojenie s virtualizáciou
Kontajnerový engine môže bežať priamo na fyzickom serveri alebo vo virtuálnom serveri, čo umožňuje flexibilitu (napr. v cloude).

Komunikácia v sieťach

- Kontajnery v tom istom hostovi → lokálna sieť
- Kontajnery medzi hostmi → overlay sieť (virtuálna nadstavba nad existujúcimi sieťami)

Použitie
Tieto pojmy sú základom pre ďalšie časti kurzu, pretože všetky pokročilé vzory a technológie (napr. Kubernetes) stavajú práve na týchto definíciách.

3. POROZUMENIE KONTAJNEROM

Táto lekcia sa hlbšie zameriava na samotné kontajnery, ich vlastnosti, funkcie a využitie v rámci architektúry distribuovaných riešení.

Kontajner a jeho obsah

- Kontajner môže obsahovať akýkoľvek typ softvérového programu, najčastejšie však aplikácie alebo služby.
- Služba (service) sa typicky vyznačuje tým, že poskytuje verejné API a je určená na vyvolávanie inými programami.
- Aplikácia môže byť samostatná alebo súčasť väčšieho distribuovaného riešenia.

Kontajnery a aplikácie/služby

- Aj aplikácia môže vystavovať API, hoci nie je navrhnutá ako služba.
- Ak aplikácia nemá API, niekedy sa pridáva pre lepšiu komunikáciu s ostatnými komponentmi v kontajneri.
- Kontajnery môžu hostovať aj iné softvérové prvky, napr. databázy.

Spôsob hostovania

- Jeden kontajner → jeden softvérový program
- Viacero kontajnerov → každý hostuje inú časť aplikácie
- Jeden kontajner môže hosťovať aj viacero súvisiacich softvérových programov, ak sú navzájom previazané

Pod a skupiny kontajnerov

- Pod umožňuje zoskupiť viac kontajnerov, ktoré patria k jednej aplikácii a potrebujú zdieľať IP adresu alebo úložisko
- Kontajnery v pod-e môžu komunikovať cez inter-process communication mechanizmy (napr. shared memory)
- Pod tiež môže poskytovať zdieľané úložisko (volume), kam ukladajú logy alebo konfiguračné súbory
- Aj keď kontajnery v pod-e zdieľajú infraštruktúru, stále sú navzájom izolované

Inštancie a klastry kontajnerov

- Rovnaký kontajner s rovnakou aplikáciou môže byť spustený viackrát ako tzv. inštancie (replicas)
- Klastry kontajnerov predstavujú množinu predinicializovaných kontajnerov, pripravených na rýchle spustenie
- Takéto klastry podporujú vysokú výkonnosť, pretože vedia rýchlo reagovať na špičku
- Klastry môžu byť dynamicky škálované podľa potreby (auto-scaling)

Orchestrace kontajnerov

- Kontajnerové platformy poskytujú tzv. orchestrátory, ktoré riadia nasadzovanie kontajnerov na základe definovaných workflow
- Orchestrátor (napr. Kubernetes) pracuje s balíčkom (package), ktorý obsahuje popis inštancie a inštrukcie na jej nasadenie
- Pred nasadením optimalizátor (deployment optimizer) vyberie vhodný hostiteľský uzol
- Orchestrátor následne kontajner spustí a monitoruje jeho stav

Kontajnerové siete

- Kontajnerová sieť umožňuje komunikáciu medzi kontajnermi v rámci riešenia
- Existujú dva hlavné typy:
  - **Host network** (kontajnery na jednom hostiteľovi)
  - **Overlay network** (kontajnery na viacerých hostiteľoch)
- Kontajnerová sieť sa dá konfigurovať tak, aby umožňovala komunikáciu s externými systémami (napr. databázami mimo kontajnera)

Adresovanie kontajnerov

- Každý kontajner dostáva vlastnú IP adresu alebo virtuálne adresovanie
- Administrátor môže určiť, do akej siete bude kontajner priradený
- Ak to nie je špecifikované, kontajnerizácia ho zaradí do predvolenej siete

3.1. ZHRNUTIE

- Kontajnery sú veľmi flexibilné, ľahko replikovateľné a dobre sa orchestrujú
- Používajú sa v moderných microservices architektúrach
- Spolu s orchestrátormi ako Kubernetes tvoria základ škálovateľných cloudových riešení

4. POROZUMENIE KONTAJNEROVÝM IMG - OBRAZ

Táto lekcia sa detailne venuje kontajnerovým obrazom, ich typom, vrstvám a spôsobu vytvárania.

Kontajnerový obraz

- Predstavuje základ všetkých kontajnerov.
- Obsahuje všetky súbory, knižnice a závislosti potrebné na vytvorenie a spustenie kontajnera.
- Kontajnerový obraz je nemenný (immutable) — ak je potrebná zmena, treba vytvoriť nový obraz.

Typy kontajnerových obrazov

- **Základné (base) obrazy**
  - slúžia ako šablóna pre vytváranie ďalších, prispôsobených obrazov
  - často obsahujú minimálne OS komponenty a knižnice
- **Prispôsobené (customized) obrazy**
  - vznikajú na základe základného obrazu a pridávajú ďalšie vrstvy, napr. aplikácie, konfiguračné súbory

Zaujímavosť je, že aj prispôsobený obraz sa môže stať základom pre ďalšie obrazy.

Nemennosť (immutability)

- Raz vytvorený obraz sa nemení.
- Ak je potrebné niečo zmeniť, vytvorí sa nový build súbor a z neho nový obraz.
- Každý obraz má unikátny identifikátor (image key), ktorý ho odlišuje od iných.

Abstrakcia operačného systému

- Obraz zvyčajne abstrahuje len tie časti OS, ktoré aplikácia skutočne potrebuje (napr. knižnice, utility).
- **Jadro (kernel)** operačného systému nie je súčasťou obrazu — kontajner ho využíva zo samotného hostiteľa prostredníctvom kontajnerového engine.
- Táto architektúra zabezpečuje prenositeľnosť medzi rôznymi prostrediami, pretože jadro poskytuje engine.

Obsah nad rámec jadra

- Súčasťou obrazu môžu byť:
  - programátorské knižnice
  - kompilátory
  - rôzne systémové knižnice
  - šifrovacie platformy
  - monitorovacie nástroje

  - konfiguračné súbory
  - lokalizačné programy
- Tým vzniká optimalizované prostredie špeciálne pre danú aplikáciu.

Build súbor (container build file)

- Konfiguračný súbor, ktorý určuje, čo sa má zahrnúť do obrazu
- Obsahuje:
  - referenciu na základný obraz
  - zoznam pridaných súborov alebo knižníc
  - konfiguráciu siete, do ktorej bude kontajner patriť
- Syntax build súboru závisí od konkrétneho container engine

Vrstvy obrazu (image layers)

- Každý obraz sa skladá z viacerých vrstiev, kde každá vrstva predstavuje jeden krok v build súbore
- Všetky vrstvy okrem tej poslednej sú read-only
- Najnižšia vrstva je základný obraz, ďalšie vrstvy ho rozširujú (napr. aplikácia samotná)
- Tento prístup umožňuje efektívnu opätovnú použiteľnosť základných vrstiev

Proces vytvárania obrazu

- Administrátor pripraví build súbor
- Container engine načíta základný obraz z image registry
- Engine podľa build súboru pridá požadované vrstvy a vytvorí nový obraz
- Nový obraz sa uloží do repozitára (lokálneho alebo verejného)

4.1. ZHRNUTIE

- Kontajnerový obraz je strojovo čitateľná, nemenná jednotka obsahujúca všetko potrebné na beh aplikácie
- Vďaka vrstvám a build súborom sú kontajnery rýchle, efektívne a prenositeľné
- Základné a prispôsobené obrazy tvoria základ ekosystému kontajnerizácie

5. MECHANIZMY KONTAJNERIZÁCIE

V tejto lekcii sa vysvetľujú technologické mechanizmy, ktoré stoja za fungovaním kontajnerov a kontajnerových prostredí.

Prehľad mechanizmov

- Kontajnerový engine
- Kontajnerový obraz
- Image registry
- Kontajnerový klaster
- Package (balíček)
- Package repository
- Kontajnerový orchestrátor
- Deployment optimizer
- Kontajnerový scheduler

Tieto mechanizmy tvoria súčasť väčšiny moderných kontajnerových platforiem.

Kontajnerový engine

- Základná komponenta celej kontajnerizácie
- Vytvára a spúšťa kontajnery na základe kontajnerových obrazov
- Spracováva build súbory
- Príkladom je Docker Engine

Kontajnerový obraz

- Predstavuje šablónu pre vytvorenie kontajnera
- Ukladá sa v image registry
- Je nemenný a opisuje prostredie, v ktorom sa aplikácia spustí

Image registry

- Centrálne úložisko kontajnerových obrazov
- Môže byť verejné (napr. Docker Hub) alebo súkromné
- Sprístupňuje obrazy pre ďalšie použitie a verzovanie

Kontajnerový klaster

- Súbor vopred pripravených kontajnerových inštancií
- Umožňuje rýchlu reakciu na nárast dopytu
- Slúži na zabezpečenie vysokej dostupnosti a škálovania

Package (balíček)

- Konfiguračný súbor, ktorý obsahuje inštrukcie a workflow pre nasadenie
- Definuje, ako sa majú kontajnery v rámci riešenia spustiť
- Spravidla obsahuje informácie o poradí spúšťania, umiestnení a konfiguráciách

Package repository

- Úložisko pre balíčky
- Podporuje správu verzií celých riešení
- Umožňuje opakovane používať deployment konfigurácie

Kontajnerový orchestrátor

- Automatizuje nasadzovanie kontajnerov na hostiteľoch
- Zabezpečuje monitorovanie ich stavu
- Spravuje ich lifecycle (napr. Kubernetes, Docker Swarm)

Deployment optimizer

- Monitoruje dostupné hostiteľské prostredia
- Vyberá optimálne miesto pre spustenie kontajnera alebo podu
- Sleduje výkon, politiku affinity/anti-affinity a ďalšie kritériá

Kontajnerový scheduler

- Plánuje a riadi dávkové úlohy
- Vie ich spúšťať paralelne alebo sekvenčne
- V niektorých prípadoch využíva nástroje ako cron alebo plánovače operačného systému

5.1. ZHRNUTIE

Mechanizmy popísané vyššie sú základom každého moderného kontajnerizačného ekosystému. Spolupracujú tak, aby kontajnery boli efektívne, spoľahlivé a ľahko riaditeľné vo veľkom rozsahu, s dôrazom na automatizáciu a škálovateľnosť.

6. VZORY MULTI-KONTAJNEROVÝCH ARCHITEKTÚR

V tejto lekcii sa vysvetľujú základné návrhové vzory, ktoré využívajú viac kontajnerov spoločne, aby sa zabezpečila funkčnosť aplikácií v distribuovanom prostredí.

Prečo multi-kontajnerové vzory?

- Väčšina moderných aplikácií má viacero častí: hlavná logika, utility, správa komunikácie.
- Ak by boli všetky tieto funkcie vložené do jedného kontajnera, zhoršila by sa údržba, škálovanie aj spoľahlivosť.
- Multi-kontajnerové vzory umožňujú tieto funkcie oddeliť a priradiť do špecializovaných kontajnerov.

Základné multi-kontajnerové vzory:

- **Sidecar Container**
  - rieši problém, keď aplikácia spracúva nielen biznis logiku, ale aj generickú utility logiku
  - utility funkcia sa vyčlení do vedľajšieho kontajnera (sidecar), aby sa hlavná aplikácia mohla sústrediť len na svoju biznis úlohu
  - sidecar býva v rovnakom pod-e
- **Adapter Container**
  - ak aplikácia musí prevádzať dáta do špeciálneho formátu pre iné externé systémy, adapter kontajner túto konverziu preberie
  - hlavná aplikácia sa tým nemusí zaťažovať a ostáva nezávislá
  - jeden adapter kontajner môže slúžiť viacerým spotrebiteľom dát
- **Ambassador Container**
  - ak aplikácia musí komunikovať so špecifickými externými systémami (napr. cez špeciálne protokoly), ambassador kontajner prevezme túto komunikáciu
  - opäť sa tým oddelí biznis logika aplikácie od detailov komunikácie
  - ambassador môže riešiť aj bezpečnostné prvky (autentifikácia, šifrovanie)

Použitie týchto vzorov spoločne

- V jednej aplikácii môžu byť nasadené všetky tri typy sekundárnych kontajnerov (sidecar, adapter, ambassador), podľa potreby.
- Každý rieši inú pomocnú úlohu, pričom hlavný kontajner zostáva zameraný len na spracovanie jadra aplikácie.
- Kombináciou vzorov sa zlepšuje flexibilita a udržiavateľnosť riešenia.

Výhody multi-kontajnerových vzorov

- Lepšia modularita
- Jednoduchšie nasadzovanie a verzovanie
- Jasné oddelenie zodpovedností
- Vyššia spoľahlivosť (napr. zlyhanie sidecar nemá priamy vplyv na biznis logiku)

6.1. ZHRNUTIE

Multi-kontajnerové návrhové vzory podporujú budovanie mikroslužbových riešení a celkovo moderných cloud-native architektúr, kde je potrebné flexibilné, škálovateľné a jednoducho spravovateľné prostredie.

7. ADMINISTRATION PATTERNS

7.1. ÚVOD

Administratívne vzory v kontajnerizácii sú navrhnuté tak, aby riešili problémy spojené so správou, prevádzkou a spoľahlivosťou kontajnerových prostredí. Cieľom je zaistiť konzistenciu, stabilitu a predvídateľnosť kontajnerov počas ich životného cyklu, čo znižuje riziká chýb, bezpečnostných incidentov alebo nepredvídaného správania.

Táto lekcia definuje 7 základných administratívnych vzorov:

1. **Container Image Integrity**
2. **Container Runtime Immutability**
3. **Container Chain**
4. **Scheduled Container**
5. **Volatile Configuration**
6. **Init Container**
7. **Observing Container**

7.2. 1. CONTAINER IMAGE INTEGRITY

Problém
Kontajnerové obrazy môžu byť počas nahrávania do registra chybne aktualizované, neoverené, zastarané alebo dokonca infikované škodlivým kódom. Keď iní správcovia tieto obrazy použijú, nasadené kontajnery môžu byť ohrozené.

Riešenie
Zavádza sa kontrolný mechanizmus na overenie a validáciu obrazov ešte pred ich uložením do registra. Každý overený obraz sa digitálne podpíše (napr. pomocou kryptografických kľúčov), aby sa zaručila jeho integrita a autenticita.

Aplikácia
Kontajnerový engine odlišuje overené obrazy od neoverených a automaticky priraďuje značky (tags), aby bolo zrejmé, ktoré obrazy sú vhodné na produkciu. Tento proces je podporený buď vstavanými funkciami enginu, alebo externým systémom na správu kľúčov.

7.3. 2. CONTAINER RUNTIME IMMUTABILITY

Problém
Ak administrátor zmení runtime prostredie v kontajneri (napr. konfiguračné súbory alebo základný runtime), môže to viesť k nestabilite aplikácie a zníženej bezpečnosti.

Riešenie
Runtime vrstva kontajnera sa nastaví ako *read-only*, čím sa zabezpečí jej nemennosť počas behu.

Aplikácia
Kontajnerový build file definuje nemenné vrstvy (napr. frameworky, runtime balíky), zatiaľ čo aplikačná logika sa môže umiestniť do zapisovateľnej vrstvy. Tým sa umožní flexibilita, ale runtime ostáva chránený.

7.4. 3. CONTAINER CHAIN

Problém
Niektoré kontajnerové aplikácie závisia na presnom poradí spúšťania a vypínania rôznych služieb (napr. databázy, API brány). Ak sa nespustia v správnom poradí, môžu zlyhať.

Riešenie
Používa sa *pod* s definovaným reťazcom spúšťania (chain), ktorý garantuje sekvenciu zapnutia/vypnutia.

Aplikácia
Administrátor cez orchestrátor definuje poradie v rámci *chain configuration* a orchestrátor zabezpečí, aby sa všetky komponenty spúšťali presne tak, ako vyžaduje aplikácia.

7.5. 4. SCHEDULED CONTAINER

Problém
Niektoré programy musia byť spustené len v určitých časoch alebo podľa konkrétneho harmonogramu (napr. dávkové úlohy), čo v štandardnom kontajneri nebýva podporované.

Riešenie
Využíva sa plánovací systém (napr. cron alebo Windows Scheduler), ktorý vie spúšťať kontajnery podľa definovaného plánu.

Aplikácia
Administrátor nakonfiguruje harmonogram v konfiguračnom súbore, ktorý sa spracuje buď cez hostiteľský systém alebo priamo cez kontajnerový orchestrátor.

7.6. 5. VOLATILE CONFIGURATION

Problém
Pri škálovaní kontajnerových služieb (in/out, up/down) môže byť konfigurácia kontajnera nedostatočná alebo zastaraná, čo spôsobuje chyby pri behu.

Riešenie
Kontajner je vybavený monitorom, ktorý priebežne sleduje runtime stav a dynamicky aktualizuje konfiguráciu tak, aby reagovala na meniace sa potreby služby.

Aplikácia
Používa sa systém monitor + konfiguračný zapisovač, ktorý aktualizuje konfiguračné súbory v reálnom čase podľa signálov o potrebe škálovania.

7.7. 6. INIT CONTAINER

Problém
Ak má hlavná aplikácia závislosti na iných utilitách, ktoré musia bežať pred jej spustením, môže dôjsť k problémom, ak sa tieto utility nespustia načas alebo spoľahlivo.

Riešenie
Použijú sa *init* kontajnery, ktoré sa spustia pred hlavnou aplikáciou a zabezpečia, že všetky prípravné úlohy sú vykonané korektne.

Aplikácia
Orchestrátor spustí init kontajnery podľa definovaného poradia, a až po ich úspešnom dokončení spustí hlavný kontajner aplikácie.

7.8. 7. OBSERVING CONTAINER

Problém
Kontajner síce vie, že aplikácia v ňom beží, ale nemusí vedieť, či funguje správne (napr. aplikácia beží, ale neodpovedá na požiadavky).

Riešenie
Kontajner implementuje kanál na sledovanie zdravia aplikácie, napríklad cez API, ktoré pravidelne odpovedá na „heartbeat“ dotazy kontajnerového enginu.

Aplikácia
Aplikačný kód sa rozšíri o podporu pre health-check API. Kontajnerový engine tak dokáže rozlíšiť, či je služba naozaj funkčná, a v prípade problémov podniknúť kroky ako reštart alebo eskalácia.

7.9. ZHRNUTIE

V tejto lekcii sú popísané základné administratívne vzory pre stabilitu, spoľahlivosť a predvídateľnosť prevádzky kontajnerov v prostredí orchestrácie. Hlavnou myšlienkou je minimalizovať riziko nekonzistencie, nesprávnych aktualizácií alebo zlého spustenia aplikácií v produkcii. Každý vzor rieši špecifický problém — od integrity obrazu, cez stabilitu runtime, až po plánovanie, monitorovanie a závislosti.

8. SCALABILITY PATTERNS

8.1. ÚVOD

Škálovacie vzory sú kľúčovým nástrojom na zaistenie, že kontajnerizované aplikácie dokážu pružne reagovať na meniace sa požiadavky používateľov a prevádzky. Tieto vzory pomáhajú spravovať výkon, dostupnosť a efektívne využitie prostriedkov v dynamických clusteroch. Ich cieľom je predchádzať preťaženiu infraštruktúry, minimalizovať neefektívne využívanie prostriedkov a zabezpečiť hladkú prevádzku aplikácií v prostrediach s vysokou dynamikou.

Táto lekcia definuje 4 základné škálovacie vzory:

1. **Automated Pod Assignment**
2. **Container Elasticity**
3. **Demanding Daemons**
4. **Finite Containers**

8.2. 1. AUTOMATED POD ASSIGNMENT

Problém
V prostredí clusterov sa môže stať, že pods (skupiny kontajnerov) sú nesprávne priradené hostom. Ak sa náročné pods nasadia na slabý host, preťažia ho, čo vedie k nedostatku výkonu. Naopak, ak sa nenáročné pods nasadia na silný server, dôjde k jeho plytvaniu.

Riešenie
Zavádza sa systém na automatické priraďovanie pods k vhodným hostom. Berie do úvahy požiadavky každého podu a dostupnú kapacitu hostov v clustri.

Aplikácia
Vzor používa tzv. *deployment optimizer*, ktorý v reálnom čase analyzuje požiadavky podov a kapacitu hostiteľov. Následne orchestrátor nasadzuje pods podľa optimalizovaného rozhodnutia. Tento systém tiež priebežne monitoruje, či sa pods nemajú premiestniť na vhodnejší host, ak sa menia podmienky (napr. vyššie zaťaženie).

8.3. 2. CONTAINER ELASTICITY

Problém
Pri kontajnerových aplikáciách so silnými špičkami zaťaženia môže dôjsť k ich nedostupnosti alebo k spomaleniu, ak infraštruktúra nie je schopná pružne reagovať na rastúci dopyt.

Riešenie
Zavádza sa horizontálne škálovanie kontajnerov – systém automaticky vytvára viac inštancií kontajnerov podľa aktuálneho dopytu.

Aplikácia
Mechanizmus kontajnerového clusteru umožňuje mať viaceré inštancie tej istej služby rozložené na viacerých hostoch. Orchestrátor na základe monitoringu dopytu koordinuje ich nasadenie aj ukončenie, čím reaguje na meniace sa zaťaženie. Ak dopyt klesne, orchestrátor môže inštancie automaticky vypnúť, aby šetril prostriedky.

8.4. 3. DEMANDING DAEMONS

Problém
Niektoré aplikácie potrebujú špecifické runtime daemony alebo podporné služby počas celej svojej životnosti. Ak nie je dostatok týchto zdrojov, program môže zlyhať.

Riešenie
Používa sa kombinácia horizontálneho aj vertikálneho škálovania, aby sa podom a ich daemon službám vždy zabezpečilo dostatočné množstvo prostriedkov.

Aplikácia
Cluster a orchestrátor zabezpečia, aby sa pods mohli škálovať horizontálne (napr. presunúť sa na iný host) aj vertikálne (zvýšenie výkonu hosta). Navyše sa využíva *deployment optimizer*, ktorý sleduje výkon a rozhoduje o tom, kedy pods premiestniť alebo posilniť ich výkon. Tento vzor sa uplatňuje hlavne v prípadoch, keď bežné horizontálne škálovanie už nestačí.

8.5. 4. FINITE CONTAINERS

Problém
Niektoré kontajnery majú byť aktívne len určitý čas alebo len dovtedy, kým splnia konkrétnu úlohu. Ak ostanú spustené príliš dlho, môžu zbytočne zaberať zdroje, prípadne dokonca spôsobovať konflikty s inými službami.

Riešenie
Zavádza sa systém, ktorý kontroluje a obmedzuje dobu existencie kontajnera.

Aplikácia
Administrátor definuje v konfiguračnom súbore podu presné parametre trvania — napríklad maximálnu dĺžku behu alebo podmienky, kedy má byť kontajner ukončený. Orchestrátor tieto parametre spracuje a priebežne sleduje, či už doba uplynula alebo bola úloha splnená. Po splnení podmienok orchestrátor kontajner automaticky ukončí.

8.6. ZHRNUTIE

Škálovacie vzory v tejto lekcii riešia problém prispôsobenia sa dynamickým požiadavkám prevádzky v prostredí kontajnerov. Umožňujú zabezpečiť dostatok výkonu v špičkách, predísť plytvaniu prostriedkov a zvyšujú celkovú efektivitu nasadenia aplikácií v kontajneroch. Ich jadrom je schopnosť orchestrátora reagovať na reálne podmienky — cez optimalizované priraďovanie, horizontálne aj vertikálne škálovanie, až po kontrolu trvania kontajnerov.

9. HIGH PERFORMANCE & RELIABILITY PATTERNS

9.1. ÚVOD

Vysoký výkon a spoľahlivosť patria medzi najdôležitejšie faktory prevádzky kontajnerových riešení. Kontajnery musia zvládať nároky na dostupnosť, rýchlosť odozvy a predchádzanie zlyhaniam. V tejto kapitole sú preto predstavené štyri návrhové vzory, ktoré pomáhajú optimalizovať výkon a zvýšiť odolnosť voči poruchám. Tieto vzory umožňujú konštruovať robustné a vysoko dostupné služby, ktoré dokážu fungovať aj v prípade výpadkov časti infraštruktúry.

Lekcia pokrýva štyri hlavné vzory:

1. **Leader Node Election**
2. **Single Host Multi-Containers**
3. **Graceful Response**
4. **Container Failover**

9.2. 1. LEADER NODE ELECTION

Problém
Niektoré úlohy vyžadujú súčinnosť viacerých inštancií tej istej služby. Ak nie je určený koordinátor (tzv. líder), môže dôjsť k nesprávnemu spracovaniu dát, duplikáciám alebo kolíziám, pretože jednotlivé inštancie sa nevedia navzájom dohodnúť.

Riešenie
Jeden z aktívnych uzlov je zvolený ako líder, ktorý koordinuje činnosť ostatných a zabezpečuje rozdelenie úloh medzi nimi.

Aplikácia
Zvyčajne sa používa špecializovaný systém (napr. Apache Kafka alebo Zookeeper), ktorý umožní voľbu lídra podľa dohodnutého algoritmu. Tento systém je často umiestnený v samostatnom kontajneri, aby sa oddelila jeho funkcia od ostatných služieb. Orchestrátor riadi voľbu lídra pri zmene prostredia alebo po výpadku uzla a zabezpečuje, aby vždy existoval jeden hlavný koordinátor úloh.

9.3. 2. SINGLE HOST MULTI-CONTAINERS

Problém
Ak sa má zložená služba skladať z viacerých menších mikroslužieb, môže pri ich komunikácii vzniknúť oneskorenie, pretože sa navzájom volajú cez sieť. To je neefektívne pri vysokých požiadavkách na výkon.

Riešenie
Jednotlivé mikroslužby sa rozdelia do samostatných kontajnerov, ale nasadia sa na ten istý fyzický alebo virtuálny host. Zdieľajú tak rovnakú infraštruktúru, ale zároveň sú logicky oddelené, čo umožňuje ich individuálne riadenie.

Aplikácia
Používa sa takzvaný service compositor, ktorý definuje logiku zloženia služby. Orchestrátor nasadí jednotlivé kontajnery na jeden host, pričom pomocou balíčkovacieho mechanizmu

(package mechanism) zabezpečí, že celý súbor potrebných kontajnerov sa nasadí spoločne a na vhodnom výkonnom prostredí. Takto sa dosiahne vysoká rýchlosť komunikácie medzi službami a súčasne možnosť samostatného škálovania jednotlivých častí.

9.4. 3. GRACEFUL RESPONSE

Problém
Keď kontajner musí byť vypnutý (napríklad kvôli aktualizácii alebo pri škálovaní), môže sa stať, že hostená aplikácia práve spracováva dáta. Ak sa kontajner ukončí náhle, dôjde k poškodeniu transakcií alebo strate dát.

Riešenie
Kontajner pred vypnutím vyšle signál aplikácii, aby sa mohla korektne ukončiť, uložiť stav a uvoľniť zdroje.

Aplikácia
Aplikácia v kontajneri musí podporovať štandardizované rozhranie (napríklad REST API) na komunikáciu s kontajnerovým enginom. Kontajner je nastavený tak, aby odoslal notifikáciu o blížiacom sa ukončení a sledoval odpoveď aplikácie. Až po potvrdení, že aplikácia bezpečne ukončila svoju činnosť, kontajner engine kontajner definitívne vypne. V prípade enterprise riešení sa odporúča tieto komunikácie zaznamenávať do logov pre audit.

9.5. 4. CONTAINER FAILOVER

Problém
Ak host, na ktorom beží kontajner, zlyhá, aplikácia sa stane nedostupnou a môže spôsobiť výpadok celého riešenia.

Riešenie
Použijú sa repliky kontajnerov nasadené na viacerých hostoch. Ak jeden host zlyhá, orchestrátor okamžite aktivuje náhradnú inštanciu na inom hoste.

Aplikácia
Kontajner cluster definuje skupinu replík (napríklad 2 alebo 3) a orchestrátor spolu s deployment optimizerom riadi, kde sa majú nachádzať. V prípade výpadku je pripravený failover mechanizmus, ktorý automaticky prepne prevádzku na iný host, čím sa zachová dostupnosť služby. Administrátor nastaví v konfiguračných súboroch požiadavku na počet replík a orchestrátor zabezpečí, aby boli vždy v aktívnom stave rozmiestnené v clustri.

9.6. ZHRNUTIE

Táto lekcia zdôrazňuje potrebu vysokého výkonu a spoľahlivosti kontajnerových aplikácií. Vzory ako voľba lídra, agregácia mikroslužieb na jednom hoste, riadené ukončenie aplikácií a failover mechanizmy tvoria základ robustných kontajnerových architektúr, ktoré odolajú aj neočakávaným udalostiam. Vďaka týmto návrhovým vzorom je možné budovať kontajnerové riešenia, ktoré zvládnu veľké objemy požiadaviek a sú pripravené na rýchlu obnovu pri poruchách.

Ak chceš, rád hneď spracujem aj **Lesson 4: Security Patterns** — mám pokračovať?

10. SECURITY PATTERNS

10.1. ÚVOD

Bezpečnosť patrí medzi najkritickejšie aspekty prevádzky kontajnerizovaných riešení. Kontajnery prinášajú výhody ako rýchle nasadenie, izoláciu a flexibilitu, ale zároveň prinášajú nové riziká spojené so zdieľaním hostiteľského operačného systému a rýchlym šírením zmien v prostredí. Cieľom tejto lekcie je popísať vzory, ktoré znižujú riziko kompromitácie kontajnerov, umožňujú kontrolu izolácie a podporujú bezpečné pripojenia.

Lekcia zahŕňa tri základné bezpečnostné vzory:

1. **Multi-Container Isolation Control**
2. **Secure Container Connection**
3. **Multiple Hosts**

10.2. 1. MULTI-CONTAINER ISOLATION CONTROL

Problém
Pri nasadzovaní viacerých kontajnerov na spoločný host môžu niektoré služby vyžadovať špecifickú úroveň izolácie — napríklad z dôvodov výkonu, licencií alebo bezpečnostných regulácií. Ak sa tieto požiadavky nedodržia, môžu nastať konflikty alebo dokonca nedostupnosť služieb.

Príklad 1
Kontajnery A a B majú vysokú vzájomnú previazanosť a potrebujú byť na tom istom fyzickom hoste kvôli rýchlej komunikácii. Ak by orchestrátor počas vyrovnávania záťaže presunul kontajner B na iný host, zníži sa výkon a môže dôjsť k problémom s prevádzkou.

Príklad 2
Naopak, kontajnery A a B reprezentujú redundantné inštancie kritickej služby a z hľadiska dostupnosti musia byť vždy na odlišných hostoch. Ak sa dostanú na jeden host a ten zlyhá, obe inštancie budú nedostupné.

Riešenie
Použije sa riadiaci mechanizmus, ktorý konfiguruje tzv. affinity a anti-affinity pravidlá. Tie určujú, ktoré kontajnery musia byť spolu (prípadne nesmú byť spolu) na rovnakom hoste. Tento mechanizmus kontroluje a audituje dodržiavanie pravidiel, spúšťa alarmy pri porušení a zabezpečuje, že sa počas dynamických zmien v clustri pravidlá neporušia.

Aplikácia
Affinity pravidlá umožnia spoluumiestnenie kontajnerov, ktoré musia koexistovať. Anti-affinity pravidlá garantujú, že dôležité inštancie sa nikdy nestretnú na jednom hoste. Orchestrátor aj deployment optimizer tieto pravidlá implementujú a uplatňujú ich pri každom plánovaní presunu alebo škálovania kontajnerov. Administrátor môže vopred testovať tzv. „what-if“ scenáre, aby preveril, či nastavené politiky zvládnu rôzne výpadky alebo preťaženia.

10.3. 2. SECURE CONTAINER CONNECTION

Problém
Nie všetky kontajnerizované aplikácie podporujú moderné bezpečné protokoly ako HTTPS

alebo TLS. Ak takéto kontajnery prijímajú aj nezabezpečené požiadavky, môžu sa stať terčom útokov alebo úniku dát.

Riešenie
Do kontajnera sa pridá dodatočná komponenta, takzvaný sidecar, ktorá dokáže filtrovať prichádzajúce požiadavky a povolí len tie, ktoré sú cez bezpečné kanály. Nezabezpečené pripojenia sú odmietnuté.

Aplikácia
Sidecar môže validovať napríklad HTTPS požiadavky pomocou certifikátov od externých autorít. Ak nie je možné zabezpečenie priamo v aplikácii, tento bezpečnostný proces sa presunie do sidecar komponenty, ktorá je v tom istom kontajneri alebo v samostatnom kontajneri vedľa. Týmto spôsobom sa zachová bezpečnosť, aj keď samotná aplikácia protokol HTTPS nepodporuje. Administrátor definuje pravidlá validácie a sidecar ich vykonáva, pričom zaznamenáva prípadné neúspešné pokusy do logov.

10.4. 3. MULTIPLE HOSTS

Problém
Kontajnery zdieľajú spoločné jadro hostiteľského operačného systému. Ak sa útočník dostane do jedného kontajnera a získal by prístup k systému, môže napadnúť celý host a všetky jeho kontajnery. To predstavuje závažné riziko pre celý ekosystém.

Riešenie
Kontajnery sa nasadzujú nie priamo na fyzický host, ale na virtualizovaný host alebo na samostatné fyzické servery, čím sa zníži dosah prípadného útoku.

Aplikácia
Najčastejšie sa využíva Type 1 virtualizácia, ktorá umožní bežať kontajnery v samostatnom virtuálnom prostredí bez tradičného operačného systému hostiteľa. Ak dôjde k napadnutiu jedného virtuálneho hosta, ostatné virtuálne hosty zostanú nedotknuté. V cloudových prostrediach sa tento prístup používa často, pretože umožňuje oddeliť kontajnery rôznych zákazníkov. Alternatívou je nasadenie každého kontajnera na vlastný fyzický server, čo síce zvyšuje náklady a znižuje úroveň konsolidácie, ale maximalizuje izoláciu a minimalizuje vplyv prípadného útoku.

10.5. ZHRNUTIE

Bezpečnostné vzory uvedené v tejto lekcii predstavujú základné stavebné prvky odolných kontajnerových prostredí. Pomáhajú chrániť nielen samotné aplikácie, ale aj infraštruktúru, na ktorej sú prevádzkované. Použitie affinity a anti-affinity pravidiel bráni nebezpečným kombináciám umiestnenia kontajnerov, sidecar komponenty zvyšujú ochranu komunikácie a virtualizácia alebo oddelené hosty izolujú kontajnery tak, aby sa prípadný útok nemohol šíriť ďalej.

ZOZNAM ZDROJOV

[1] McKendrick R. , (2020). Mastering Docker, Fourth Edition, Packt Publishing, p. 568, ISBN: 9781839216572.

[2] Poulton N. , (2023). The Kubernetes Book, Nielsen Book Services, p. 355, ISBN: 9781916585003.

[3] Kubernetes, Kubernetes. Available online: kubernetes.io

[4] Docker, Docker, Available online: docs.docker.com