Praxe II: PodSizer

Návrh a realizace informačního nástroje pro srozumitelnější práci s paměťovým nastavením JVM aplikací

Jan Weber

Závěrečný popis projektu a reflexe

V rámci předmětu Praxe II jsem pracoval na projektu PodSizer. Jedná se o desktopovou aplikaci vytvořenou ve frameworku Fyne, která slouží jako nástroj pro návrh a ověření paměťového nastavení JVM aplikací běžících v Kubernetes. Projekt přímo navazuje na můj studijní cíl ze Studijního semináře II, ale v této části už pro mě nebylo hlavní samotné učení frameworku. Cílem praxe bylo vytvořit konkrétní výstup, který bude mít praktické využití a zároveň bude odpovídat mému zájmu o propojování designu informačních služeb s technickou praxí site reliability engineera.

Téma jsem si vybral proto, že s podobným problémem se v pracovním prostředí reálně setkávám. Při provozu aplikací v Kubernetes je potřeba správně nastavovat systémové prostředky tak, aby aplikace byly stabilní, efektivní a bezpečně provozovatelné. U JVM aplikací je to složitější v tom, že paměť není tvořena pouze jednou částí. Do celkové spotřeby vstupuje heap, metaspace, code cache, direct memory, native libraries, thread stack, typ a režie garbage collectoru a dodatečná rezerva pro výsledný request a limit containeru. Přesto se v praxi často stává, že člověk při výpočtu potřebné paměti uvažuje hlavně nad jediným segmentem, tedy heapem, a ostatní části paměti zůstávají méně viditelné a opomenuté.

Z hlediska designu informačních služeb pro mě nebyl hlavním problémem samotný výpočet, ale způsob, jak s těmito informacemi člověk pracuje. Hodnoty jako heap, metaspace, direct memory nebo thread stacks existují v dokumentaci, metrikách i zkušenosti seniornějších kolegů, ale pro méně zkušeného uživatele nemusí být snadné pochopit jejich vztah k výslednému nastavení kontejneru. PodSizer jsem proto vnímal jako nástroj, který má pomoci převést rozptýlené technické znalosti do přehlednějšího pracovního postupu.

Cílovou skupinou pro první verzi byli hlavně lidé, kteří se pohybují v provozu aplikací v Kubernetes a potřebují rychle odhadnout nebo ověřit paměťové nastavení JVM aplikace. Může jít o site reliability engineery, DevOps specialisty, vývojáře nebo juniornější kolegy, kteří se s tímto tématem teprve seznamují. Právě u nich mi dávalo smysl nesoustředit se pouze na přesný výsledek, ale také na vysvětlení, z čeho výsledek vzniká.

PodSizer jsem proto navrhoval jako pomůcku, která umožní jednotlivé části JVM paměti zadat, přepočítat a zobrazit v jednom přehledném prostředí. Nechtěl jsem vytvořit pouze kalkulačku, která vrátí jedno číslo, ale aplikaci, která uživateli ukáže, z čeho se výsledná hodnota skládá. To je důležité hlavně ve chvíli, kdy člověk potřebuje vysvětlit nebo obhájit, proč má mít aplikace právě takto nastavené requests a limits.

Na projektu jsem pracoval přibližně tři týdny. Šlo o intenzivní práci, protože jsem současně řešil výpočetní logiku, strukturu aplikace, rozhraní ve Fyne a první podobu vizualizace. Výsledkem není finální produkt, ale funkční prototyp, který už bylo možné použít v praxi. Současná verze ověřuje hlavní princip aplikace, tedy zadání hodnot, výpočet doporučeného nastavení, vygenerování JVM flags a vizuální zobrazení paměťových regionů.

První část práce spočívala v ujasnění rozsahu. Původní téma by se dalo rozšiřovat mnoha směry, například směrem k CPU, více kontejnerům v jednom podu, načítání hodnot z clusteru nebo pokročilejším scénářům podle typu aplikace. Pro první verzi jsem ale rozsah záměrně zúžil na paměť JVM aplikací. Chtěl jsem vytvořit něco, co bude dostatečně malé na realizaci v rámci praxe, ale zároveň dostatečně konkrétní, aby to mělo reálný užitek.

Při práci jsem se snažil postupovat designérsky, ne jen programátorsky. Pomohly mi nejen základní principy dvojitého diamantu, ale také IBM Design Thinkingu, se kterými jsem se setkal v jednom z kurzů v rámci samostudia v předmětu MOOC: Learning Online. V praxi to pro mě znamenalo nejprve si rozšířit porozumění problému, potom zúžit rozsah plánovaných funkcí na řešitelnou první verzi, vytvořit prototyp, ověřit jej a na základě zjištění pokračovat další iterací. Tento postup mi pomáhal i v samotném vývoji, protože mě nutil nepřidávat funkce bez rozmyslu a přemýšlet, co je pro první verzi opravdu důležité.

Aplikace v současné podobě umožňuje zadávat hodnoty pro jednotlivé části JVM paměti a typ garbage collectoru. Na základě těchto vstupů vypočítá celkovou paměť JVM, celkovou paměť JVM včetně native částí, GC overhead a jako hlavní výstup zobrazí doporučené velikosti requestu a limitu kontejneru. Zároveň vygeneruje odpovídající JVM flags. Uživatel tak dostává nejen číselný výsledek, ale i konkrétní hodnoty, se kterými může dále pracovat při nastavování aplikace.

Při návrhu výstupů jsem se snažil rozlišit, co uživatel potřebuje vidět hned a co je spíše podpůrná informace. Jako hlavní výstup jsem proto zvolil doporučené hodnoty requestu a limitu kontejneru, protože s nimi uživatel přímo pracuje při konfiguraci aplikace. JVM flags jsou doplňujícím praktickým výstupem a vizualizace slouží hlavně k pochopení souvislostí.

Důležitou součástí projektu byla vizualizace paměťových regionů. Ta zobrazuje, jakou část celku zabírají jednotlivé segmenty paměti. Smyslem této vizualizace není estetické doplnění výpočtu, ale lepší pochopení poměrů mezi jednotlivými částmi paměti. Právě tato část podle mě nejvíce propojuje technickou a designovou stránku projektu, protože převádí složitější výpočet do podoby, kterou lze rychleji přečíst a interpretovat.

Výzvou pro mě nebylo jen vytvořit výpočetní logiku, ale také ji dostat do podoby, která bude uživatelsky srozumitelná. Uvědomil jsem si, že technicky správný výstup sám o sobě nestačí. Pokud má aplikace pomáhat v rozhodování, musí člověku ukázat vztahy mezi hodnotami tak, aby jim dokázal porozumět a dále s nimi pracovat.

Při realizaci jsem vycházel ze své pracovní zkušenosti a zároveň jsem se v rozhovorech s kolegy ptal, co by od podobného nástroje potřebovali. Nešlo o rozsáhlý výzkum, spíše o menší průběžné ověřování během práce. Aplikaci jsme zatím úspěšně použili celkem tři lidé, včetně mě, při výpočtech a úpravách nastavení aplikačních komponent v pracovním prostředí. To pro mě bylo důležité ověření základní funkčnosti a zároveň potvrzení, že projekt dává smysl. Opíral jsem se i o vlastní zkušenost s tím, že podobnou aplikaci jsem nikde na webu nenašel.

Při práci jsem si ujasňoval problém, cílovou situaci a rozsah první verze, potom jsem prototyp postupně upravoval podle toho, co se ukázalo při reálném použití a při rozhovorech s kolegy. Jedním z nejvýraznějších rozhodnutí byla změna podoby vizualizace. Tím, že prototypu nepředcházel detailní návrh ve Figmě, ale jen velmi jednoduchý wireframe navržený v GoodNotes na iPadu, stala se pro mě změna vizualizace důležitým momentem celého projektu. Původně jsem pracoval s horizontálním rozložením segmentů. Tento návrh dával smysl jako první představa, protože připomínal rozdělení jednoho celku na části. V praxi se ale ukázalo, že se menší segmenty špatně vejdou a některé popisky jsou hůře čitelné. Proto jsem vizualizaci přepracoval do vertikální podoby. Tím se zlepšila čitelnost a jednotlivé části dostaly více prostoru. Současně ale vím, že ani tato podoba ještě není konečná.

Původně jsem zvažoval postup přes detailnější návrh ve Figmě, ale u tohoto projektu jsem se rozhodl pro funkční prototyp přímo ve Fyne. Důvodem bylo hlavně to, že jsem framework teprve poznával a dopředu jsem nevěděl, jaké možnosti a omezení bude mít při tvorbě vlastních komponent a rozložení. Tento postup mi umožnil rychle zjišťovat, co funguje, co je obtížnější a kde je potřeba návrh upravit podle technologie. Zároveň se mi ale potvrdilo, že je důležité věci nejdříve promyslet a navrhnout mimo kód, a to i v případě týmu o velikosti jednoho člověka, protože následné změny jsou v nástroji jako Figma mnohem rychlejší než přepisování hotové implementace.

Při vývoji jsem řešil také strukturu samotné aplikace. Nechtěl jsem mít vše jen v jednom souboru bez jasného rozdělení, protože počítám s tím, že projekt může pokračovat. Proto jsem oddělil část s výpočetní logikou od části uživatelského rozhraní a snažil se pracovat s vlastními typy v Go. Vycházel jsem přitom i z menšího průzkumu open-source projektů, abych se mohl inspirovat tím, jak se podobné aplikace strukturují. I toto pro mě bylo důležitou součástí praxe, protože rozšiřitelný projekt není jen otázkou funkcí v rozhraní, ale také toho, jak je aplikace postavená uvnitř.

Za hlavní úspěch projektu považuji to, že aplikace funguje a byla reálně použita. I když není dokončená ve všech detailech, splnila svůj základní účel. Pomáhá spočítat hodnoty relevantní pro nastavení JVM aplikací v Kubernetes a zároveň ukazuje, z čeho se výsledná paměť skládá. To může být užitečné nejen při samotném nastavování resources, ale také při vysvětlování problému méně zkušeným kolegům.

Současně vnímám jasné limity aktuální verze. Z hlediska inkluzivního designu a přístupnosti aplikace ještě potřebuje poměrně velký update. Bude potřeba lépe promyslet barvy, kontrast, čitelnost popisků, práci s malými segmenty, vysvětlení hodnot a celkovou vizuální hierarchii. Fyne má v této oblasti určité limity, ale zároveň očekávám, že s novějšími verzemi frameworku přibudou další možnosti, které půjde využít. Aktuální verzi proto nevnímám jako konec projektu, ale jako funkční základ pro další práci.

V dalších iteracích chci PodSizer dále rozvíjet, zlepšit uživatelské rozhraní, zpřesnit vizualizaci a rozšířit výpočetní možnosti tak, aby aplikace lépe odpovídala reálným scénářům práce s JVM aplikacemi v Kubernetes.