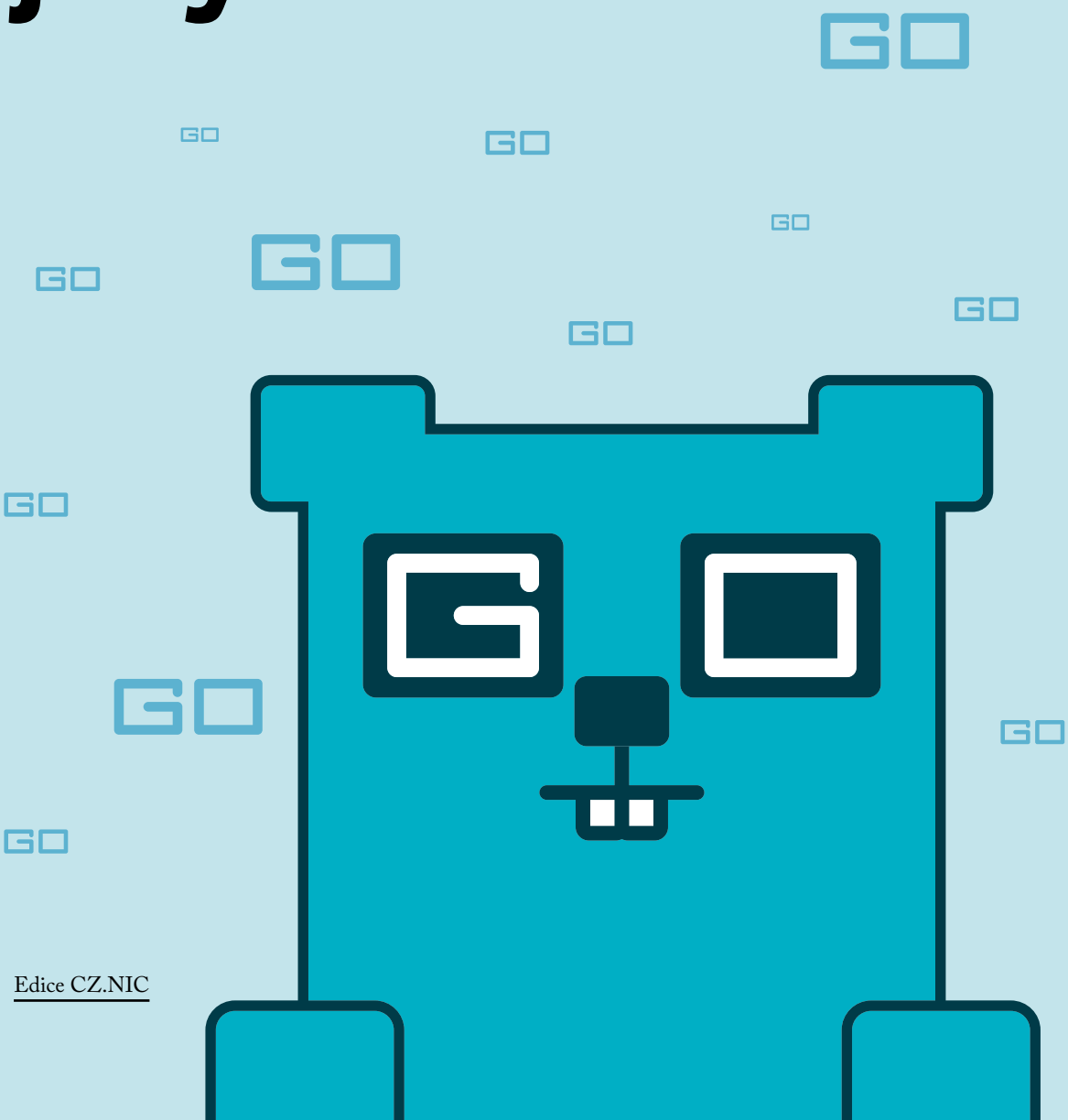
Pavel Tišnovský

Programovací jazyk GO



Edice CZ.NIC

PROGRAMOVACÍ JAZYK GO

Pavel Tišnovský

Vydavatel:
CZ.NIC, z. s. p. o.
Milešovská 5, 130 00 Praha 3
Edice CZ.NIC
www.nic.cz

1. vydání, Praha 2025
Kniha vyšla jako 34. publikace v Edici CZ.NIC.
ISBN 978-80-88168-83-6

© 2025 Pavel Tišnovský
Toto autorské dílo podléhá licenci Creative Commons BY-ND 4.0 (https://creativecommons.org/licenses/by-nd/4.0/). Dílo však může být překládáno a následně šířeno v písemné či elektronické formě, na území kteréhokoliv státu, za předpokladu, že nedojde ke změně díla a i nadále zůstane zachováno označení autora a prvního vydavatele díla, sdružení CZ.NIC, z. s. p. o. Překlad může být šířen pod licencí CC BY-ND 4.0.

Tato kniha vyšla v Edici CZ.NIC. Chcete přispět na vznik dalších? Darujte libovolnou částku na **dar.nic.cz/kniha-go**

Edice CZ.NIC je jednou z osvětových aktivit sdružení CZ.NIC, správce české národní domény.

CZ.NIC | SPRÁVCE DOMÉNY CZ

ISBN 978-80-88168-83-6

— Pavel Tišnovský

Programovací jazyk GO

— Edice CZ.NIC

Předmluva vydavatele

Předmluva vydavatele

Po knize Evoluce Pythonu, která nabídla pohled na vývoj jednoho z nejrozšířenějších programovacích jazyků, přichází autor s novým titulem zaměřeným na jazyk Go. Tentokrát se soustředí na praktický vývoj backendových aplikací, ale opět zůstává u stylu, který jde dál než jen k samotným pravidlům jazyka: snaží se postihnout širší kontext a vývojové souvislosti.

Přestože jazyk Go a Python jsou v mnoha ohledech odlišné, autor přistupuje k tématu podobně – věcně, s důrazem na srozumitelnost a využitelnost v praxi. Text není psán jako učebnice, ale spíš jako průvodce pro vývojáře, kteří hledají praktické odpovědi i prostor pro vlastní úvahy.

Jsme rádi, že můžeme tuto knihu nabídnout čtenářům, kteří se o jazyk Go zajímají nebo s ním již pracují, a zároveň si cení otevřeného pohledu, který autor k tématu přináší.

Za celý tým Edice CZ.NIC přejeme příjemné čtení.
Praha, 19. června 2025

Předmluva

Předmluva

V současnosti se mohou vývojáři setkat s celou řadou programovacích jazyků. Tyto aktivně používané programovací jazyky se od sebe v mnoha oblastech odlišují a taktéž se nasazují v různých oblastech IT. Mezi jeden z nejpopulárnějších programovacích jazyků současnosti patří i programovací jazyk nazvaný Go, s jehož možnostmi se podrobněji seznámíme v této knize. Jedná se o výkonný kompilovaný a taktéž o silně typovaný programovací jazyk, který je navržen takovým způsobem, že základy tohoto jazyka je možné se naučit doslova za několik hodin a již za přibližně týden či čtrnáct dní je v něm možné tvořit aplikace určené pro produkční nasazení. V současnosti tvoří aplikace a nástroje vyvinuté v jazyku Go jádro většiny komponent ekosystému tzv. kontejnerů. V Go je naprogramován například Kubernetes, OpenShift, Podman, Docker, Prometheus, ale i další populární nástroje, mezi které patří NATS, MinIO či NSQ.

Obsah

1 Úvod

1 Úvod

V současnosti se můžeme ve vývojářské praxi setkat s celou řadou programovacích jazyků. Tyto aktivně používané programovací jazyky se od sebe pochopitelně v mnoha oblastech odlišují a taktéž se nasazují v různých oblastech IT (některé na frontendu, další spíše na backendu, v oblasti datové analýzy atd.). Mezi jeden z nejpopulárnějších programovacích jazyků současnosti patří i programovací jazyk nazvaný Go, s jehož možnostmi se podrobněji seznámíme v této knize.

Jedná se o výkonný kompilovaný (překládaný) a taktéž o silně typovaný programovací jazyk, který je navržen takovým způsobem, že základy tohoto jazyka je možné se naučit doslova za několik hodin a již za přibližně týden či čtrnáct dní je v něm možné tvořit aplikace určené pro produkční nasazení. Taktéž se jedná o jazyk, jenž je určen pro vývoj služeb (resp. možná přesněji řečeno mikroslužeb) popř. nástrojů ovládaných z příkazového řádku. V současnosti tvoří aplikace a nástroje vyvinuté v jazyku Go jádro většiny komponent ekosystému tzv. kontejnerů (*containers*). Konkrétně to znamená, že v Go je naprogramován například Kubernetes, OpenShift, Podman, Docker, Prometheus, ale i další populární nástroje, mezi které patří NATS, Minio či NSQ.

1.1 Autoři jazyka Go

Go při svém představení vzbudil ohlas nejenom kvůli svým technickým vlastnostem, ale i díky tomu, že pochází z „dílny" Google a taktéž proto, že za jeho designem stojí mj. i dvě slavné osobnosti z oblasti informatiky: Rob „Commander" Pike a Ken Thompson.

1.2 Go nebo Golang?

Oficiální název tohoto programovacího jazyka je Go. To je ovšem poměrně nešťastně zvolené jméno, protože Go je jak název známé deskové hry, tak i běžné slovo v angličtině. To způsobuje problémy při vyhledávání dalších informací o tomto jazyku a mj. i z tohoto důvodu se poměrně často setkáme i s označením Golang (Go language). V této knize se ovšem budeme držet oficiálního jména Go.

2 Jazyk Go v současnosti

2 Jazyk Go v současnosti

V současnosti nalezneme aplikace a služby psané v jazyce Go především na serverech, tedy na *back endu*. V Go je naprogramována poměrně velká část služeb souvisejících s kontejnery, ale taktéž některé databáze (například CockroachDB), různé formy message brokerů atd. Pro zajímavost se o některých těchto technologiích alespoň ve stručnosti zmíníme.

2.1 Podman

Podman neboli *pod manager* je nástroj určený pro správu kontejnerů. Využívá knihovnu `libpod` a nabízí rozhraní (API) pro správu životního cyklu kontejnerů, jednotlivých podů, obrazů (image) i svazků. Důležité je, že rozhraní Podmanu je kompatibilní s rozhraním známého systému Docker, takže administrátoři, kteří znají Docker, mohou na Podmana snadno přejít. A podobně jako Docker nabízí *Docker Desktop* je možné v případě systému Podman využívat *Podman Desktop*.

2.2 Kubernetes

Kubernetes je systém určený pro takzvanou *orchestraci* virtualizace prováděnou na úrovni operačního systému. Původně jej vyvinula společnost Google (ta stála i za vývojem samotného Go) a jako podřízené nástroje podporuje například známý Docker ale i výše zmíněný Podman. S využitím Kubernetes je možné spouštět vybrané a nakonfigurované úlohy v takzvaných kontejnerech. Díky tomu se velmi často používá v cloudových řešeních, datových centrech atd.

2.3 MinIO

Velmi úspěšným projektem napsaným v jazyce Go je MinIO. Jedná se o sadu několika služeb a nástrojů, které uživatelům poskytují distribuované datové úložiště určené pro ukládání obecných (nestrukturovaných) dat. Typicky se jedná o soubory používané v oblasti AI (Artifical Intelligence) a ML (Machine Learning), ovšem kromě těchto populárních (a vlastně do značné míry i módních) oblastí IT je pochopitelně možné službu MinIO použít i pro ukládání logů, souborů, k nimž je zapotřebí rychle přistupovat z mnoha různých, mnohdy vzájemně vzdálených oblastí (zde využijeme možnost distribuovaného systému), jako centrální úložiště dokumentů, obrázků, videí, pochopitelně i obrazů souborových systémů pro Docker apod. MinIO dosahuje velmi slušné rychlosti přístupu k datům (při vhodně nadimenzované síti, která je většinou limitujícím faktorem) a mj. i díky velmi dobré stabilitě ukazuje přednosti programovacího jazyka Go, ve kterém je celý systém naprogramován.

Jedním z nejdůležitějších a v důsledku i nejpraktičtějších vlastností projektu MinIO je fakt, že se pro přístup k datům používá stejná technologie, jaká je implementována i v populární službě

Amazon S3 či možná přesněji AWS S3. To mj. znamená, že dodávaný MinIO Client SDK může sloužit jak pro přístup k datům uloženým v Miniu, tak i k datům uloženým ve cloudu na S3. Díky tomu lze například snadněji nastavit konfiguraci pro vývoj, konfiguraci CI, zajistit si možnost využití veřejného cloudu (S3) nebo naopak privátního cloudu (založeného na Miniu) atd.

2.3.1 NATS

Systém NATS patří mezi takzvané message brokery, což jsou systémy určené pro doručování, ukládání a redistribuci zpráv. NATS je poměrně úspěšný projekt, jenž je vyvinut právě v programovacím jazyce Go, což mu do značné míry zajišťuje stabilitu i škálovatelnost. To jsou ostatně vlastnosti, které u message brokerů většinou očekáváme. NATS v současnosti poměrně zdárně konkuruje dalším message brokerům, mezi které patří například RabbitMQ, ActiveMQ (Artemis), IBM MQ atd.

2.3.2 NATS JetStream

Součástí projektu NATS je i technologie nazvaná NATS JetStream. S jejím využitím lze realizovat takzvaný streaming zpráv s podobnými vlastnostmi, jaké nalezneme u „konkurenčního" projektu Apache Kafka. NATS JetStream používá klasický systém (server) NATS, který navíc doplňuje o takzvaný storage, tj. o technologii určenou pro ukládání zpráv (někdy nazývaných i záznamy – record) do perzistentního úložiště, kterým může být relační databáze či soubor resp. skupina souborů.

2.3.3 NSQ

V jazyce Go je vytvořen i další message broker nazvaný nsq. Ten se skládá z několika samostatně spouštěných uzlů, které mezi sebou vhodným způsobem komunikují. Výhodou a jednou ze základních vlastností nsq je fakt, že způsob jeho návrhu počítá s tím, že jednotlivé uzly mohou být po nějakou dobu nedostupné a přitom systém jako celek by měl být stále použitelný. Jedná se tedy o architekturu vhodnou pro nasazení do dnešního hybridní prostředí se systémy běžícími v cloudu, lokálně v kontejneru, na dedikovaných serverech atd.

2.3.4 FZF

Jedním z nástrojů psaných v Go, které se těší poměrně značné popularitě, je utilita nazvaná *fzf* neboli „command-line fuzzy finder". Základní funkce *fzf* je ve skutečnosti velmi jednoduchá a vlastně i typicky „unixovská". Nástroj *fzf* totiž na svém standardním vstupu (*stdin*) získá libovolný seznam, který následně zobrazí a nechá uživatele vybrat jednu položku z tohoto seznamu. Výsledek následně předá na svůj standardní výstup (*stdout*) pro další zpracování. To pravděpodobně nezní

moc bombasticky ani příliš užitečně, ovšem samotný výběr položky ze seznamu je plně interaktivní a podporuje takzvaný „fuzzy“ výběr (snadno použitelný je i pro ty uživatele, kteří neznají regulární výrazy). A především – největší síla *fzf* spočívá v tom, že tento nástroj je možné kombinovat s vybranými standardními příkazy popř. dalšími utilitami, všude tam, kde je nutné vybrat hodnotu ze seznamu (platného například v nějakém kontextu).

3 Go a další programovací jazyky

3 Go a další programovací jazyky

Programovací jazyk Go a jeho základní knihovny byly původně navrženy takovým způsobem, aby se v Go daly efektivně psát různé síťové aplikace, které by ideálně neměly trpět potenciálními výkonnostními problémy nebo bezpečnostními dírami typu „buffer overflow" atd. Právě z toho důvodu, že síťové a serverové aplikace musí v typické konfiguraci obsluhovat velké množství souběžně prováděných požadavků, byly do jazyka Go přidány již v úvodní kapitole zmíněné *gorutiny* a *kanály*. A nutno říci, že Go se v této oblasti skutečně velmi často používá. Taktéž se předpokládalo, že se jazyk Go bude používat pro vývoj těch aplikací, které jsou v současnosti psány v jazycích C či C++ (možná i v Rustu). Tento přechod sice skutečně částečně nastal, ovšem prozatím ne v příliš velké míře (a to se bavíme o použití v serverových a desktopových aplikacích, protože v oblasti mikrořadičů se s jazykem Go sice laboruje, ale především u relativně výkonných MCU typicky založených na čipech Cortex-M, nikoli na malých osmibitových a šestnáctibitových mikrořadičích).

3.1 Od Pythonu ke Go?

Několik let po představení tohoto jazyka však nastala zajímavější situace, o níž jsme se opět krátce zmínili v úvodu – programovací jazyk Go a jeho relativní snadnost použití a poskytované možnosti objevili vývojáři, kteří psali své síťové a speciálně pak webové aplikace v Pythonu, Ruby či JavaScriptu/TypeScriptu. Přepis takových aplikací do jazyka Go mnohdy znamenal řádový a někdy i dvouřádový (tedy 100× a více) nárůst výkonu těchto aplikací. Do jisté míry je nárůst výkonu způsobem překladem do nativního kódu, ovšem nesmíme zapomenout na gorutiny, které nejsou (na rozdíl od klasických vláken) příliš náročné na paměť, takže se můžeme setkat s aplikacemi, v nichž bez větších problémů běží stovky či dokonce tisíce gorutin (což si ostatně ještě ukážeme v prakticky zaměřené části této knihy).

3.2 Go z pohledu vývojáře

Go je staticky typovaný programovací jazyk a programy, které jsou v něm napsány, jsou kompilovány (překládány) do nativního kódu (někdy se setkáme s přesnějším označením *strojový kód* neboli *machine code*). Použití statických datových typů vede k tvorbě bezpečnějšího kódu a společně s výše uvedeným konceptem překladu do strojového kódu jsou výsledné aplikace velmi rychlé, zejména v porovnání s aplikacemi naprogramovanými například v Pythonu či v dalších interpretovaných jazycích.

Programovací jazyk Go navíc programátorům nabízí i koncept výše zmíněných *gorutin* a kanálů používaných při komunikaci mezi gorutinami. Díky tomu lze velmi snadno a především bezpečně psát aplikace, jejichž části běží souběžně (*concurrently*) či dokonce paralelně.

3.3 Go není „pouze" vylepšený programovací jazyk C

Mezi autory návrhu i první implementace programovacího jazyka Go patří mj. i Ken Thompson. Když si uvědomíme, že Ken vytvořil i (dnes prehistorický) programovací jazyk B, který je předchůdcem slavného jazyka C, mohlo by se zdát, že Go bude pouhým vylepšením programovacího jazyka C (například nějaká obdoba jazyka C++). Je sice pravda, že Go alespoň částečně z programovacího jazyka C skutečně vychází, ovšem z hlediska syntaxe jsou si mnohem bližší například jazyky C a Java, než C a Go.

Tvůrci programovacího jazyka Go se navíc snažili poučit se z některých problematických rysů jazyků C a taktéž C++. Z tohoto důvodu navrhli Go takovým způsobem, aby byla jeho syntaxe jednodušší a snáze pochopitelná (sami se po prohlédnutí zdrojových kódů můžete rozhodnout, do jaké míry se to splnilo). Samotný jazyk je velmi konzervativní a někteří kritici dokonce (s nadsázkou) tvrdí, že zcela ignoruje všechny novinky, které se na poli programovacích jazyků za posledních dvacet let objevily.

3.4 Automatická správa paměti

Navíc se tvůrci jazyka Go zaměřili i na další typické problémy, s nimiž se musí vypořádat prakticky všichni programátoři používající jazyky C, C++ a vlastně i uživatelé Rustu (ti nepřímo): jedná se o problematiku správy paměti a uvolňování zdrojů (*resources*) a v neposlední řadě i o podporu pro paralelní běh částí programů, například částí komunikujících přes síť, pracujících se souborovým systémem atd. Z tohoto důvodu byly přímo do jazyka Go přidány dvě důležité technologie: automatický správce paměti a již výše zmíněné *gorutiny* (*goroutines*). S oběma těmito technologiemi se podrobněji seznámíme v navazujících kapitolách.

Poznámka: na tomto místě je vhodné si uvědomit, že použití automatického správce paměti přenáší část problémů (které by jinak musel řešit programátor) do takzvaného runtime*, což se může negativně projevit na výkonu aplikace (zpomalení aplikace, větší nároky na kapacitu haldy, musí se počítat i s okamžiky, kdy je aplikace pozastavena apod.). Vlivem správce paměti na výkon se budeme zabývat v samostatné kapitole, takže zde jen bez dalších důkazů uveďme, že automatický správce paměti implementovaný v jazyku Go nemá větší negativní vliv na dobu startu aplikací (ostatně, nemusí se například spouštět ani virtuální stroj ani inicializovat interpret), spotřeba paměti je však průměrně větší, než v případě C/C++, ale na druhou stranu menší, než je tomu při porovnání s obdobnými aplikacemi psanými v Javě.*

3.5 Minimalisticky pojatý programovací jazyk

Některé technologie naopak do jazyka Go přidány *nebyly*, a to z toho důvodu, aby byl jazyk snadno použitelný a navíc i snadno implementovatelný. Právě fakt, že se v Go *zpočátku* neobjevily dnes již široce akceptované technologie, jako je obsluha výjimek či práce s generickými datovými typy, byl poměrně často kritizován; ovšem Go se postupně mění a přidávají se do něj další vlastnosti. Ovšem tyto nové vlastnosti jsou přidávány relativně pomalu a poměrně konzervativním způsobem, což je podle názoru autora této knihy v této oblasti jen dobře. Poslední velkou změnou je přidání podpory pro generické datové typy.

4 Instalace překladače jazyka Go i dalších podpůrných nástrojů

4 Instalace překladače jazyka Go i dalších podpůrných nástrojů

I když existují různá webová „pískoviště“, ve kterých je možné si vyzkoušet základy programovacího jazyka Go, je pro výuku tohoto jazyka výhodnější přímá instalace všech základních nástrojů Go, tj. jak vlastního překladače, tak i nástrojů pro naformátování zdrojových kódů, instalaci modulů, prohlížení nápovědy atd. Instalace Go se pochopitelně liší podle toho, jaký operační systém používáte. Z tohoto důvodu je popis instalace rozdělen do několika částí – instalace Go na systému Microsoft Windows a instalace na vybraných distribucích Linuxu.

4.1 Instalace v operačním systému Microsoft Windows

Instalace nástrojů jazyka Go pro operační systém Microsoft Windows je snadná a poměrně rychlá. Postačuje přejít na stránku https://go.dev/. Kromě dalších informací zde nalezneme tlačítka *Get Started* a *Download*. Stiskneme tlačítko *Download* a po přechodu na další stránku v sekci nazvané *Featured downloads* (zhruba v polovině obrazovky) nalezneme i instalátor nástrojů Go pro systém Microsoft Windows. V době psaní této knihy se jedná o soubor `go1.24.2.windows-amd64.msi`, ovšem číslo verze se pochopitelně bude postupně zvyšovat. Tento soubor je nutné stáhnout a spustit. Další kroky instalace jsou snadné – instalátor se spustí, ponechají se všechna výchozí nastavení a tlačítkem `Next` se postupně instalace dokončí. Následně si spusťte konzoli (postačuje `cmd.exe`) a v ní napište příkaz:

```
go
```

V případě, že instalace proběhla v pořádku, měla by se vypsat nápověda k nástrojům jazyka Go.

4.2 Instalace v operačním systému Linux

Instalace jazyka Go v operačním systému Linux se liší v závislosti na tom, jakou konkrétní distribuci používáte a jaký má tato distribuce systém pro správu balíčků. V současnosti se používá několik správců balíčků, ovšem nejčastěji se pravděpodobně setkáte se systémem nazvaným *yum* (*Yellowdog Updater Modified*) resp. jeho následovníkem *dnf*, nebo s nástrojem *apt* (*Advanced Package Tool*). Z tohoto důvodu byly pro účely popisu instalace Go vybrány dva typy distribucí Linuxu. Do první skupiny patří Fedora, do skupiny druhé pak Ubuntu a Linux Mint.

4.2.1 Instalace do distribuce Fedora

Popišme si nyní instalaci základních nástrojů programovacího jazyka Go do distribuce Fedora. Instalace bude na všech třech posledních vydáních Fedory prakticky totožná, takže se zaměřme

na Fedoru 40, pro niž byl vytvořen balíček s Go verze 1.23.8 (což je v současnosti stabilní verze tohoto jazyka). Instalaci provedeme buď přímo z terminálu s přihlášeným superuživatelem (rootem) příkazem:

```
dnf install golang
```

Alternativně samozřejmě můžeme použít instalaci s využitím nástroje `sudo`, pokud má ovšem přihlášený uživatel příslušná práva:

```
sudo dnf install golang
```

V obou případech by měla instalace proběhnout prakticky totožným způsobem:

```
================================================================================================
 Package                         Architecture    Version              Repository        Size
================================================================================================
Installing:
 golang                          x86_64          1.23.8-1.fc40        updates          667 k
Installing dependencies:
 go-filesystem                  x86_64          3.5.0-1.fc40         fedora           8.8 k
 golang-bin                      x86_64          1.23.8-1.fc40        updates           27 M
 golang-src                      noarch          1.23.8-1.fc40        updates           13 M
 libserf                         x86_64          1.3.10-5.fc40        fedora            59 k
 subversion-libs                 x86_64          1.14.4-1.fc40        updates          1.5 M
Installing weak dependencies:
 mercurial                       x86_64          6.7.4-3.fc40         updates          6.4 M
 python3-fb-re2                  x86_64          1.0.7-15.fc40        fedora            23 k
 subversion                      x86_64          1.14.4-1.fc40        updates          1.0 M

Transaction Summary
================================================================================================
Install  9 Packages

Total download size: 50 M
Installed size: 239 M
Is this ok [y/N]:
```

Po odpovědi „Y“ se instalace rozběhne a pochopitelně se nijak zásadně neliší od instalace jakéhokoli jiného balíčku:

```
Downloading Packages:
(1/9): go-filesystem-3.5.0-1.fc40.x86_64.rpm                                  113 kB/s | 8.8 kB
00:00
...
...
...
Running transaction
  Running scriptlet: golang-1.23.8-1.fc40.x86_64                               1/1
...
...
...
Installed:
  go-filesystem-3.5.0-1.fc40.x86_64     golang-1.23.8-1.fc40.x86_64        golang-
bin-1.23.8-1.fc40.x86_64
  golang-src-1.23.8-1.fc40.noarch       libserf-1.3.10-5.fc40.x86_64
mercurial-6.7.4-3.fc40.x86_64
  python3-fb-re2-1.0.7-15.fc40.x86_64   subversion-1.14.4-1.fc40.x86_64    subversion-
libs-1.14.4-1.fc40.x86_64

Complete!
```

Povšimněte si, že se po nainstalování rozbalí přibližně 239 MB dat, což je sice poměrně velká hodnota, ovšem je vhodné si uvědomit, že se kromě samotného překladače jazyka Go nainstalují i další podpůrné nástroje (formátovač zdrojového kódu atd.), celá základní knihovna Go, zdrojové kódy základní knihovny (ty slouží pro čtení dokumentace) atd. Samotný ekosystém jazyka Go se všemi svými nástroji vyžaduje necelých 300 MB diskového prostoru.

4.2.2 Ubuntu a Mint

V distribucích Ubuntu a Mint se jako správce balíčků používá nástroj *apt* (*Advanced Package Tool*). Instalaci Go můžeme provést buď jako administrátor (root) příkazem:

```
apt install golang
```

nebo lze, jako na Fedoře, použít příkaz `sudo`, ovšem za předpokladu, že jste součástí skupiny uživatelů, kteří mohou získat administrátorská práva:

```
sudo apt install golang
```

Průběh instalace probíhá zhruba následovně:

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
golang-1.23 golang-1.23-doc golang-1.23-go golang-1.23-src golang-doc
golang-go golang-src
Suggested packages:
bzr | brz mercurial subversion
The following NEW packages will be installed:
golang golang-1.23 golang-1.23-doc golang-1.23-go golang-1.23-src
golang-doc
golang-go golang-src
0 upgraded, 8 newly installed, 0 to remove and 623 not upgraded.
Need to get 82,5 MB of archives.
After this operation, 438 MB of additional disk space will be used.
```

4.2.3 Lokální instalace

V případě, že nemáte práva administrátora (a ani možnost použít příkaz `sudo`) lze instalaci nástrojů jazyka Go provést i lokálně – pouze pro přihlášeného uživatele. Opět přejděte na stránku https://go.dev/, na níž nalezneme tlačítka *Get Started* a *Download*. Stiskneme tlačítko *Download* a po přechodu na další stránku v sekci nazvané *Featured downloads* nalezneme i balíček s Go pro Linux (takzvaný *tarball*). V době psaní této knihy se jedná o soubor se jménem `go1.24.2.linux-amd64.tar.gz`. Ten stáhneme a přesuneme do vhodného adresáře, například do domovského adresáře uživatele. V tomto adresáři stačí otevřít terminál (konzoli) a napsat příkaz pro rozbalení staženého *tarballu*:

```
tar xvfz go1.24.2.linux-amd64.tar.gz
```

Po rozbalení vznikne adresář nazvaný `go`. Nastavte si cestu do podadresáře `go/bin`:

```
PATH=go/bin:$PATH
```

Nyní by měl být dostupný příkaz `go` i další příkazy ekosystému jazyka Go.

4.3 Kontrola, zda je Go nainstalován korektně

Po instalaci základních nástrojů programovacího jazyka Go je vhodné si otestovat, jestli jsou tyto nástroje skutečně použitelné. Většina nabízených funkcí je dostupná přes „univerzální“ příkaz `go`. V případě, že na terminál (konzoli) zapíšeme pouze tento příkaz bez dalších doplňujících informací, měla by se vypsat krátká nápověda:

```
Go is a tool for managing Go source code.

Usage:

    go <command> [arguments]

The commands are:

    bug         start a bug report
    build       compile packages and dependencies
    clean       remove object files and cached files
    doc         show documentation for package or symbol
    env         print Go environment information
    fix         update packages to use new APIs
    fmt         gofmt (reformat) package sources
    generate    generate Go files by processing source
    get         add dependencies to current module and install them
    install     compile and install packages and dependencies
    list        list packages or modules
    mod         module maintenance
    work        workspace maintenance
    run         compile and run Go program
    test        test packages
    tool        run specified go tool
    version     print Go version
    vet         report likely mistakes in packages

Use "go help <command>" for more information about a command.

Additional help topics:

    buildconstraint build constraints
    buildmode       build modes
    c               calling between Go and C
```

```
    cache           build and test caching
    environment     environment variables
    filetype        file types
    go.mod          the go.mod file
    gopath          GOPATH environment variable
    goproxy         module proxy protocol
    importpath      import path syntax
    modules         modules, module versions, and more
    module-auth     module authentication using go.sum
    packages        package lists and patterns
    private         configuration for downloading non-public code
    testflag        testing flags
    testfunc        testing functions
    vcs             controlling version control with GOVCS

Use "go help <topic>" for more information about that topic.
```

4.3.1 Ověření verze jazyka Go

Pro začátek si vypíšeme, která verze jazyka Go je vlastně aktuálně dostupná. K tomuto účelu slouží následující varianta již výše zmíněného příkazu `go`:

```
go version
```

V případě, že jste si nainstalovali Go verze 1.23.3 na 64bitové architektuře x86-64 (známé též pod označením AMD64), měla by se na plochu terminálu vypsat následující zpráva:

```
go version go1.23.3 linux/amd64
```

Při instalaci starší verze Go se pochopitelně vypíše odlišná verze jazyka. Opět si uveďme příklad:

```
go version go1.22.1 linux/amd64
```

V této kapitole jsme si ukázali instalaci jazyka Go ve verzi 1.23 popř. 1.24. Ve skutečnosti budou všechny demonstrační příklady pracovat i ve starší verzi 1.22 (a mnohé i ve starších verzích, teoreticky i verze 1.9 atd.). Go je v tomto ohledu velmi konzervativní a i když se jazyk postupně vyvíjí, zachovává si zpětnou kompatibilitu.

4.3.2 Proměnné prostředí používané jazykem Go

V některých situacích je nutné si ověřit konfiguraci Go. Pro zobrazení všech proměnných, které ovlivňují chování Go, se používá příkaz:

```
go env
GO111MODULE=''
GOARCH='amd64'
GOBIN=''
GOCACHE='/home/ptisnovs/.cache/go-build'
GOENV='/home/ptisnovs/.config/go/env'
GOEXE=''
GOEXPERIMENT=''
GOFLAGS=''
GOHOSTARCH='amd64'
GOHOSTOS='linux'
GOINSECURE=''
GOMODCACHE='/home/ptisnovs/go/pkg/mod'
GONOPROXY=''
GONOSUMDB=''
GOOS='linux'
GOPATH='/home/ptisnovs/go'
```

Samozřejmě je možné si nechat vypsat jen jednu konkrétní hodnotu, například:

```
go env GOPATH
/home/ptisnovs/go
```