Увод. Развој веб апликација у пуном стеку. МПК пројектни образац. Контејнеризација.

Веб програмирање

Доц. др Слободан Јелић

Грађевински факултет, Универзитет у Београду
Основне академске студије
смер: Геоинформатика
1. предавање

Београд, 2. октобар 2024.

Садржај

[1] (енг. multi-container applications)

Увод

Зашто учимо веб програмирање?

Извор: https://www.techiebears.com/what-is-the-importance-of-web-development/, датум приступа: 1. октобар 2024.

Увод

Важност веб програмирања уопште

- једно од најпопуларнијих подручја рачунарског програмирања
- већина десктоп апликација одавно су замењене одговарајућим веб апликацијама
- цели системи прелазе на веб пословање (банкарски сектор, институције јавног сектора (е-Управа))
- кључне предности у односу на десктоп апликације:
  - заобилажење географских и културолошких дистанци
  - позитиван допринос развоју пословања у различитим сферама на глобалном нивоу
  - смањење трошкова и повећање ефикасности пословања
- примене у геоинформатици: приказ интерактивних карата са геолошким својствима

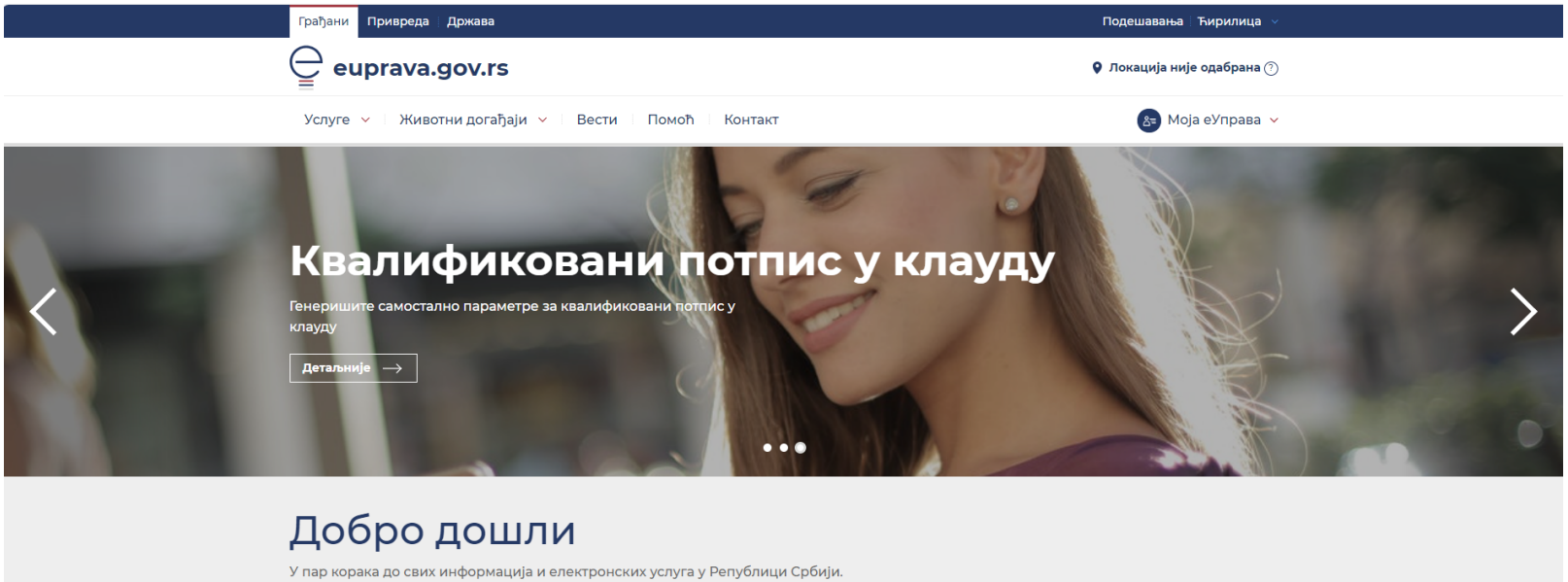
Увод

Пример: Веб шоп

Извор: `https://vectorportal.com/vector/online-shopping-vector-graphics-concept.ai/33729`, датум приступа: 1. октобар 2024.

Увод

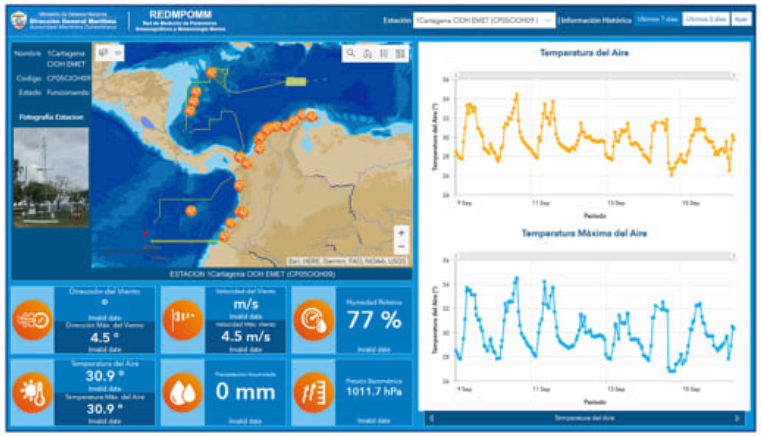
Пример: Е-управа

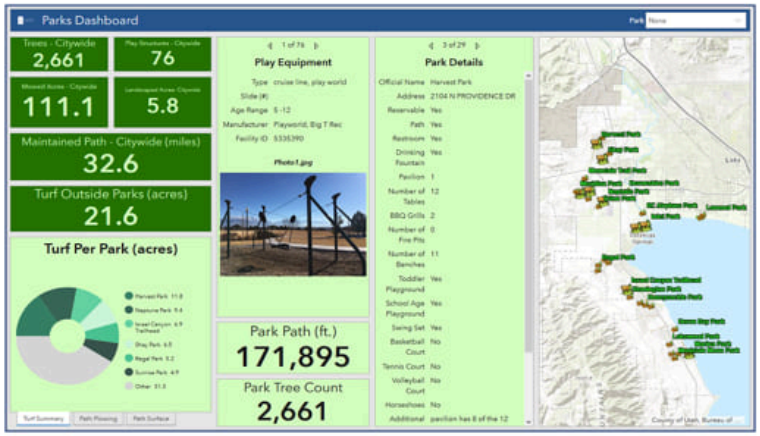


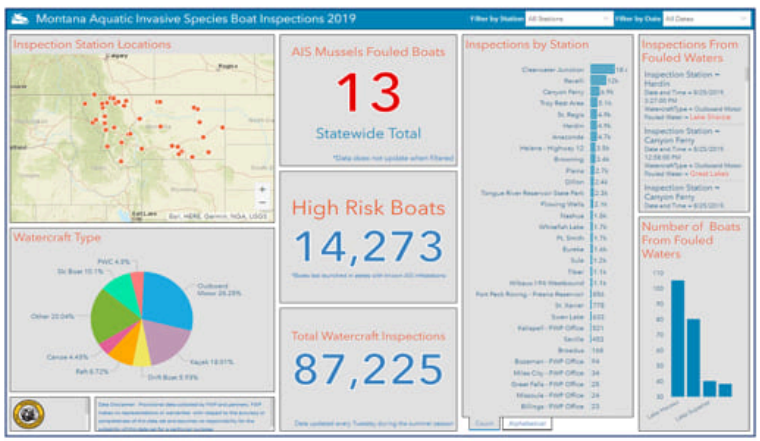
Извор: `https://euprava.gov.rs`, датум приступа: 1. октобар 2024.

Увод

Важност веб програмирања у геоинформатици







Извор: `https://www.esri.com/arcgis-blog/products/ops-dashboard/decision-support/dashboards-in-arcgis`, датум приступа: 1. октобар 2024.

Увод

Важност веб програмирања у геоинформатици

- интеграције са ГИС[6] системима
- апликације за прикупљање, чишћење, манипулацију и приказивање гео-података
- посебно се истичу веб апликације за приказивање гео-података које садрже контролне табле[7]

[6] (енг. GIS - Geographic Information System)
[7] (енг. dashboard)

Дефиниција (Развој апликација у пуном стеку)

Развој веб апликација у пуном стеку[8] представља развој **клијентске** (енг. frontent) и **серверске** (енг. backend) стране апликације користећи клијент-сервер мрежну архитектуру. Према клијент-сервер архитектури клијентска страна веб апликације извршава се на клијентским уређајима[9], а серверска на једном или више сервера. Клијентска страни задужена је за приказ података и интеракција са корисником, док се на серверској страни налази имплементација пословног процеса и база података.

[8] (енг. full stack web development)

[9] рачунари, таблети, мобилни телефони

Технологије

- на клијентској страни апликације користимо JavaScript и библиотеку ReactJS
- на серверској страни апликације користимо Python и библиотеке: Flask и SqlAlchemy
- користимо релациону базу података PostgreSQL

МПК пројектни образац

Пројектни образац: Аутономни преглед[13]

- овај образац понекад се назива и **паметан кориснички интерфејс** (енг. SMART UI)

Дефиниција (Аутономни преглед (АП))

Аутономни преглед је пројектни образац развоја апликација[10] који користи парадигме објектно оријентисаног програмирања на начин да једна класа преузима све одговорности апликације која има графички кориснички интерфејс (енг. GUI - Graphical User Interface) :

- прима догађаје које подстиче корисник[11], на пример: клик миша или притисак тастера на тастатури,
- користи алгоритам[12] да претвори догађаје ,подстакнуте од стране корисника у промене стања апликације,
- одржава релевантно стање апликације,
- изводи визуално рендеровање стања апликације.

[10] не искључиво веб апликација, него апликација уопште
[11] (енг. user driven events)
[12] или апликацијску логику
[13] (енг. Autonomous View)

МПК пројектни образац

Пројектни образац: Документ-преглед[17]

- образац решава проблеме АП обрасца
- подела одговорности на две класе: `Dokument`[14] и `Pregled`[15]

Дефиниција (Документ-Преглед (ДП))

Документ-Преглед је пројектни образац развоја апликација који користи парадигме објектно оријентисаног програмирања на начин да **две класе**, Документ и Преглед, преузимају одговорности апликације која има графички кориснички интерфејс, на следећи начин:

- **Документ** класа имплементира алгоритам или логику апликације, чува стање апликације, имплементира методе за приступ стању апликације, промену стања апликације и обавештавање других објеката о промени стања,[16]

---

[14] (енг. Document)
[15] (енг. View)
[16] није обавезно
[17] (енг. Document-View)

МПК пројектни образац

Пројектни образац: Документ-преглед[18]

- **Преглед** класа обрађује догађаје подстакнуте од стране корисника, рендерује визуализацију стања апликације, проводи операције над класом Документ и синхронизује визуализацију стања апликације када се оно мења ун класи Документ.

[18] (енг. Document-View)

МПК пројектни образац

Пројектни образац: Модел-Преглед-Контролер[19]

- улогу класе Документ преузима класа **Модел** без промене одговорности
- одговорност класе Преглед дели се на класе **Преглед** и **Контролер**

Дефиниција (Модел-Преглед-Контролер (МПК))

Модел-Преглед-Контролер је пројектни образац развоја апликација који користи парадигме објектно оријентисаног програмирања на начин да **три класе**, преузимају одговорности апликације која има графички кориснички интерфејс, на следећи начин:

- **Модел** класа имплементира логику апликације и чува стање апликације, имплементира методе за приступ стању апликације, промену стања апликације и обавештавање других објеката о промени стања **- доменска логика**
- **Преглед** класа рендерује визуализацију стања апликације **- презентацијска логика**

[19] (енг. Model View Controller)

МПК пројектни образац

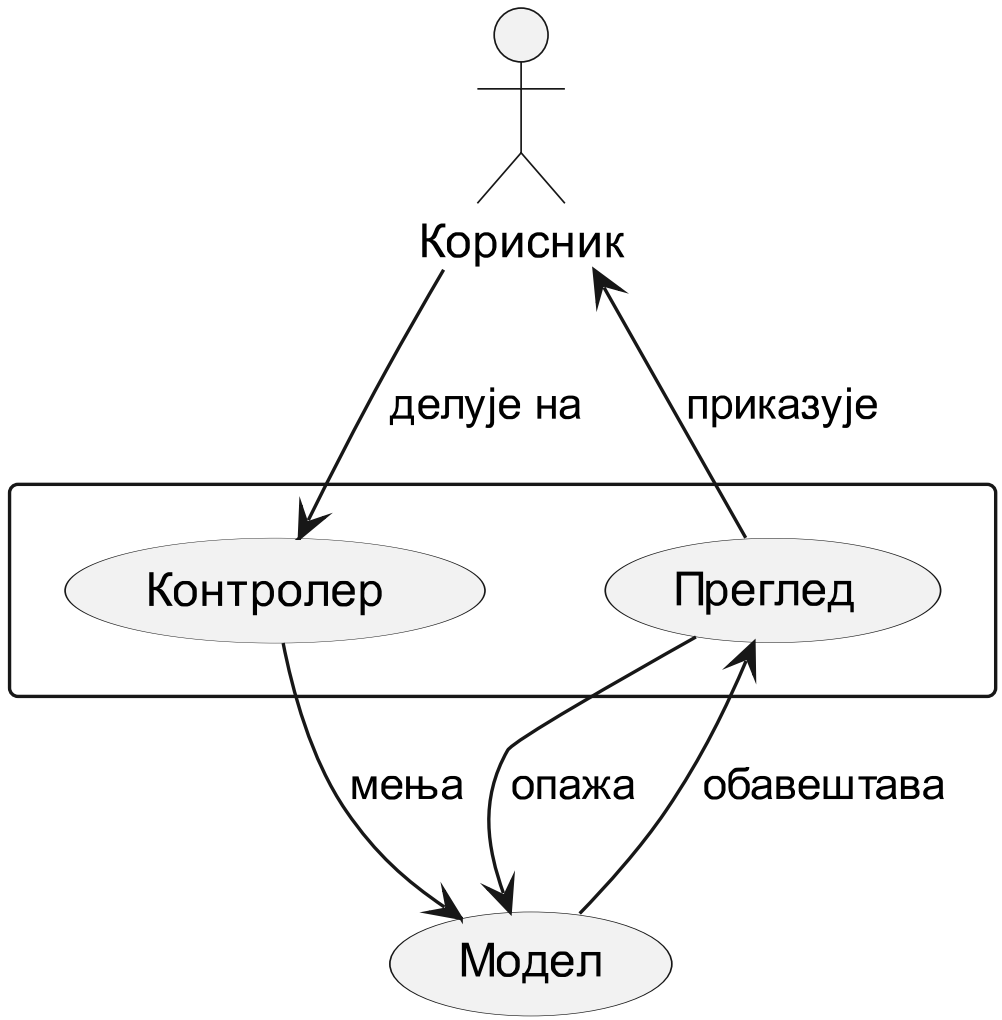
Пројектни образац: Модел-Преглед-Контролер[20]

- **Контролер -** класа обрађује догађаје подстакнуте од стране корисника на графичком корисничком интерфејсу и покреће промене на класи **Моделу -** (**апликацијска логика**)

[20] (енг. Model View Controller)

МПК пројектни образац

Пројектни образац: Модел-Преглед-Контролер[21]



[21] (енг. Model View Controller)

Контејнеризација апликације

Докер платформа и контејнери

Дефиниција (Контејнер)

Контејнер је спакована апликација у окружењу које има следећа четири својства:

- **изолација -** дефинисање изолованог окружења које је независно од домаћина (енг. host) на којем се извршава
- **потпуност -** окружење садржи све што је потребно за извршавање апликације без обзира на сервисе и апликације инсталиране на домаћину
- **преносивост -** окружење може да се дели и преноси те да се апликација у њему извршава без обзир на домаћина
- **постојаност -** окружење је увек исто на сваком домаћину

Контејнеризација апликације

Докер платформа и контејнери

- докер платформа омогућује:
  - развој апликације са свим компонентама унутар контејнера
  - извршавање више различитих сервиса/апликација у различитим контејнерима на истом домаћину
  - дистрибуција и тестирање апликације унутар контејнера
  - постављање апликације у продукцијско окружење без обзира да ли се ради о локалном серверу, провајдеру услуга ”у облаку” (као што су Google , AWS , Azure)

Контејнеризација апликације

Примене докера

- **брза** и **конзистентна** испорука апликација:
  - докери омогућују конзистентност испоруке због својства постојаности контејнера
  - присетимо се да својство постојаности осигурава да ће испоручена апликација да се извршава у идентичном окружењу без обзира на домаћина
  - брзина испоруке огледа се у значајном олакшању током развоја софтвера
  - сви програмери могу да развијају апликацију у идентичном стандардизованом развојном окружењу на својој локалној машини
  - контејнери пружају стандардизованост развојном окружења

Контејнеризација апликације

Примене докера

- значајна примена докера налази се у CI/CD (енг. Continuous Integration continuous Delivery/Deployment) процедурама:
  - докери олакшавају и тестирање апликације у тестном окружењу због својства преносивости и потпуности
  - могуће је дефинисати јединичне и интеграционе тестове унутар контејнера и поставити их у тестно окружење

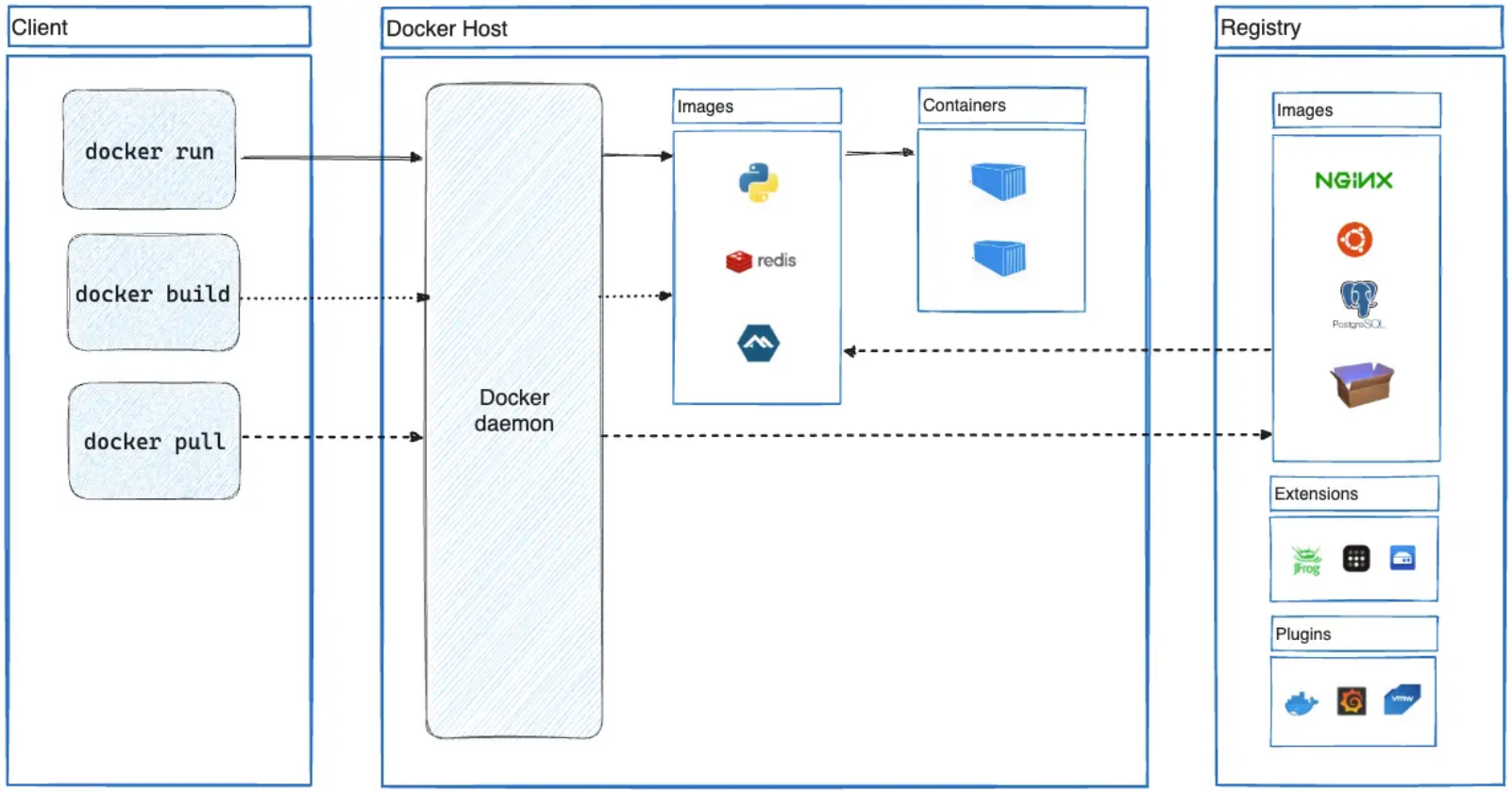
Контејнеризација апликације

Примене докера

- респонзивно постављање и скалирање апликације
  - докер платформа омогућује преносивост апликација у широком спектру различитих платформи
  - једноставна скалабилност на основу пословних захтева
- извршавање више различитих сервиса на једном домаћину
  - ценовно прихватљива алтернатива за окружење са више виртуалних машина
  - омогућује постављање много апликација које не захтевају велике ресурсе

Контејнеризација апликације

Примене докера



Архитектура докера

- клијент-сервер архитектура
- докер клијент користи Docker CLI (енг. Command Line Interface) који комуницира са докер сервером
- докер сервер је позадински процес (енг. docker daemon)
- комуникација између докер клијента и докер сервера одвија се на 3 могућа начина:
  - REST API ,
  - Unix socket ,
  - network interface .
- осим докер клијента постоји и Docker Compose који такође има улогу клијента
- користи се за контејнеризацију апликација које се састоје од више одвојених сервиса (следе микро-сервис архитектуру)

Архитектура докера

- Docker Daemon - `dockerd` - докер сервер, позадинска апликација која одговара на API захтеве докер клијената и одговорна је за управљање докер објектима као што су слике, контејнери, волумени и мреже
- Docker client - `client` - докер клијент, шаље API захтеве докер серверу `dockerd` у погледу управљања докер објектима.

Пример

Када покренемо на терминалу покренемо наредбу `docker run` онда докер клијент `docker` шаље захтев докер серверу за покретање контејнера. Докер сервер одговара том захтеву с покретањем контејнер или слањем поруке о грешци.

Архитектура докера

- Docker Desktop - апликација за оперативне системе Windows , Mac и Linux која омогућује развој и покретање контејнеризованих апликација користећи докер клијента (docker и docker compose), докер сервер и остале сервисе
- Docker Registries - регистри или репозиторијуми где се чувају слике докер контејнера, одакле могу да се повуку (енг. pull) или где могу да се гурну (енг. push) .
- постоје јавни и приватни докер репозиторијуми
- Docker Hub јер пример јавног докер репозиторијума

Докер објекти

Слике

- као што сама реч каже, слике представљају обрасце/шаблоне за изградњу контејнера
- слика се репрезентује текстуалним фајлом под називом `Dockerfile`
- унутар тог фајла дефинише се како треба да изгледа контејнер (зато кажемо да је то слика контејнера)
- углавном се ради о наредбама које се извршавају приликом изградње контејнера на основу слике

Докер објекти

Слике

- најчешће се слика дефинише из других слике додавањем посебних наредби којима се оне мењају, прилагођавају или им се додају/уклањају компоненте
- архитектура контејнера је слојевита (енг. layer) и сваком наредбом у `Dockerfile`-у се дефинише један слој
- када се промени слика, потребно је поново изградити (енг. rebuild) само оне слојеве који су се променили

Докер објекти

Контејнери

- контејнери (енг. container) су инстанце слике које могу да се покрећу и извршавају
- могу да се **креирају**, **покрећу**, **заустављају** и **бришу** користећи API наредбе докер клијента
- може да се контролише њихова мрежа и спремиште
- изоловани су од домаћина на којем се извршавају
- на њихову конфигурацију, осим слике према којој су изграђени, утичу и додатне опције које се прослеђују докер сервер помоћу докер клијент наредби у тренутку изградње
- све настале промене у стању контејнера се губе када се тај контејнер брише, осим ако нису спремљене на неко трајно спремише

Контејнери

Покретање контејнера

1. Проверити да ли је дефинисана слика контејнера у фајлу `Dockerfile`
2. Из коренског директоријума пројекта покрените следећу наредбу:

```
docker build -t welcome-to-docker .
```

3. Покрените контејнер из апликације Docker Desktop

Контејнери

Покретање контејнера

- слика се дефинише у `Dockerfile`-у
- дајемо следећи једноставан пример слике

```
# syntax=docker/dockerfile:1
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

- апликација има дефинисане пакете и изворни код у фолдеру `src`
- наш задатак је да се упознамо и објаснимо принцип изградње слике за апликацију из примера

Контејнери

Покретање контејнера

- до сада смо већ видели неке од наредби у Dockerfile -у
- `FROM` - инструкција за постављање базне слике, може да буде било која валидна слика контејнера
- `WORKDIR` - инструкција за постављање радног директоријума и односи се на наредбе: `RUN`, `CMD`, `ENTRYPOINT`, `COPY` и `ADD`; ако радни директоријум не постоји, биће креиран
- `COPY` - инструкција за копирање фајла или директоријума из локалног фајл система домаћина у изоловани систем контејнера чију слику градимо; може да садржи и додатне опције за привилегије приступа копираним фајловима/директоријумима

Контејнери

Покретање контејнера

- `RUN` - инструкција за извршавање било које наредбе у тренутку изградње слике
- `CMD` - инструкција за извршавање наредбе унутар контејнера када се слика изгради и контејнер покрене
- `EXPOSE` - инструкција за излагање порта потребног за `TCP` или `UDP` мрежну комуникацију с контејнером; углавном се излаже порт за приступ серверском делу апликације који се извршава у контејнеру
- потпуне референце инструкција за дефиницију слике: https://docs.docker.com/reference/dockerfile/

Контејнери

Покретање контејнера

- слика се изграђује покретањем следеће команде у фолдеру у којем се налази `Dockerfile`

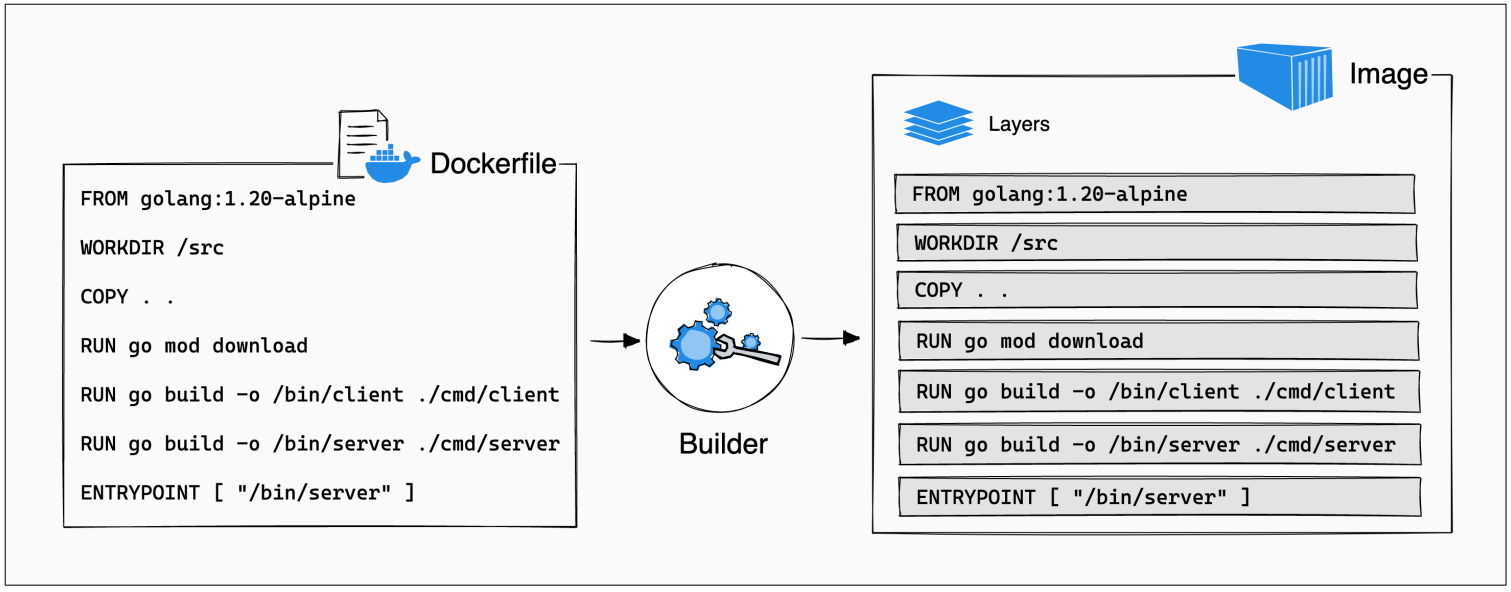
```
docker build -t getting-started .
```

- обавезни аргумент је путања или `URL` којом се дефинише контекст изградње слике; ако се наводи URL параметер, то може да буде гит репозиторијум, компресована слика (енг. pre-packaged tarball contexts) или текстуални фајл
- `-t` - назив и таг слике која ће бити изграђена; ако се наводи таг, следи се формат `nazv:tag`
- референца: https://docs.docker.com/reference/cli/docker/image/build/

Контејнери

Покретање контејнера

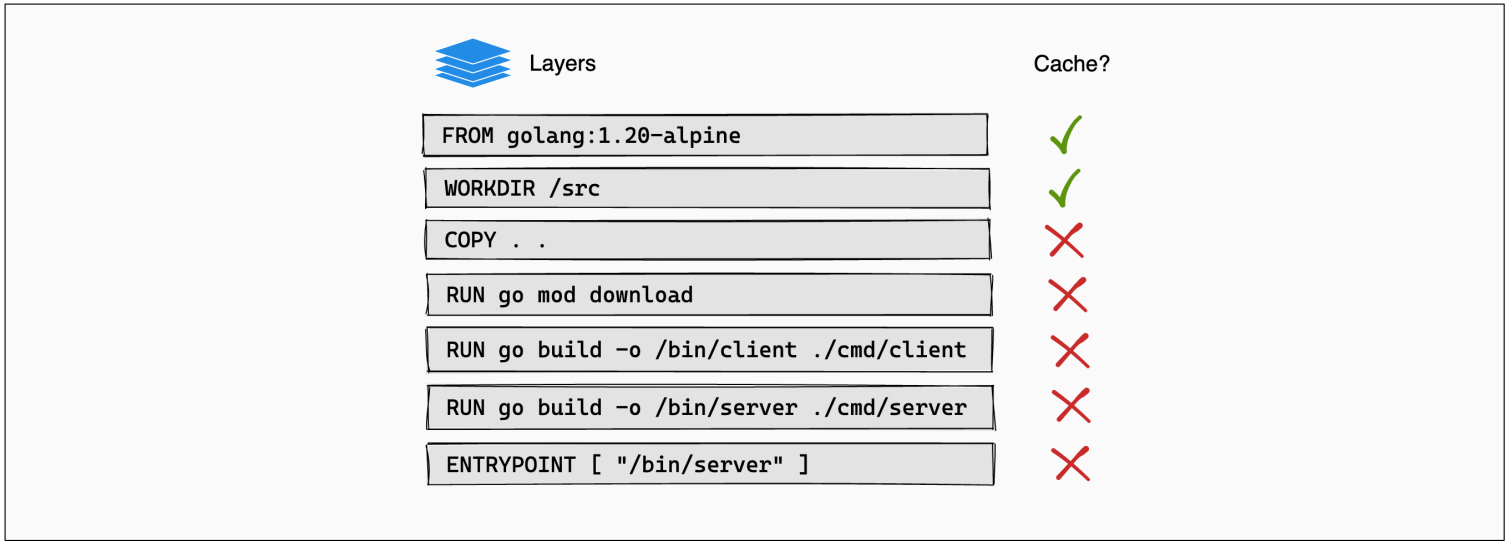
- редослед инструкција у `Dockerfile`-у је битан јер одређује редослед слојева који се изграђују
- свака инструкција одговорна је за изградњу једног слоја слике



Контејнери

Покретање контејнера

- приликом покретања изградње слике, билдер најпре покушава да искористи неизмењене слојеве из претходних изградњи (ако такви постоје) које је кеширао
- ако се промени било који од пројектних фајлова, то ће да поништи `COPY` слојеве у кешу



Контејнери

Покретање контејнера

- покретање контејнера врши се помоћу наредбе `docker run`

```
docker run -dp 127.0.0.1:3000:3000
↪  getting-started
```

- `-d` или `--detach` - извршава контејнер у позадини враћајући се на терминал домаћина
- `-p` креира мапирање порта између домаћина и контејнера према синтакси `HOST:CONTAINER`
- референца: https://docs.docker.com/reference/cli/docker/container/run/

Контејнери

Покретање контејнера

- уколико желимо да мењамо апликацију, морамо поново да изградимо контејнер
- претпоставимо да желимо да урадимо следећу промену у

```
- <p className="text-center">No items yet! Add one
↪  above!</p>
+ <p className="text-center">Josh uvek nemamo
↪  dodatih poslova na spisku.</p>
```

- уколико извршимо следеће наредбе

```
docker build -t getting-started .
docker run -dp 127.0.0.1:3000:3000 getting-started
```

добијамо поруку о грешци

Контејнери

Покретање контејнера

- следећом наредбом проверавамо који контејнери се тренутно извршавају

```
docker ps
```

- потребно је најпре зауставити контејнер из претходне верзије апликације

```
docker stop <the-container-id>
```

- а потом га и обрисати

```
docker rm <the-container-id>
```

Контејнери

Покретање контејнера

- готово код свих апликација потребно је осигурати перзистентност/трајност података
- два су разлога за то: изолованост фајл система контејнера и губитак комплетног фајл система приликом брисања контејнера
- докер пружа решење креирањем **волумена**

```
docker volume create todo-db
```

- након заустављања и брисања контејнера, можемо да покренемо нови контејнер који ћи да отвори (енг. mount) управо креирани волумен `todo-db`

```
docker run -dp 127.0.0.1:3000:3000 --mount
↪  type=volume,src=todo-db,target=/etc/todos
↪  getting-started
```

Контејнери

Покретање контејнера

- инспекција волумена

```
docker volume inspect todo-db
```

Контејнери

Покретање контејнера

- готово код свих апликација потребно је осигурати перзистентност/трајност података
- два су разлога за то: изолованост фајл система контејнера и губитак комплетног фајл система приликом брисања контејнера
- докер пружа решење креирањем **волумена**

```
docker volume create todo-db
```

- након заустављања и брисања контејнера, можемо да покренемо нови контејнер који ћи да отвори (енг. mount) управо креирани волумен `todo-db`

```
docker run -dp 127.0.0.1:3000:3000 --mount
↪  type=volume,src=todo-db,target=/etc/todos
↪  getting-started
```

Апликације са више контејнера[22]

- природан избор код развоја веб апликације пуног стека
- компоновање пуног апликацијског стека који се састоји од сервиса, волумена и мрежа
- дефиниција целог стека у једном `YAML` фајлу
- погодан за све фазе животног циклуса апликације: **развој**, **тестирање**, **стејџовање** и **продукцију**
- користимо `docker compose` команду

[22] (енг. multi-container applications)

Апликације са више контејнера[23]

Предности апликација са више контејнера

- односе се на развој, постављање и пуправљање контејнеризованих апликација:
  - **једноставна контрола** - поједностављена орекстрација и координација комплексних сервиса у једном `YAML` фајлу
  - **ефикасна колаборација** - поједностављен заједнички рад више развојних тимова
  - **брзи развој апликације** - користи се кеширани контејнери сервиса који нису промењени у току тазвоја
  - **преносивост** - могужност лаког преношења из једног окружења у друго (на пример из развојног окружења у тестно окружење, или из стејџа у продукционо окружење)
  - **активна заједница о подршка** - заједница која пружа подршку и приручнике за рад

[23] (енг. multi-container applications)

Апликације са више контејнера[24]

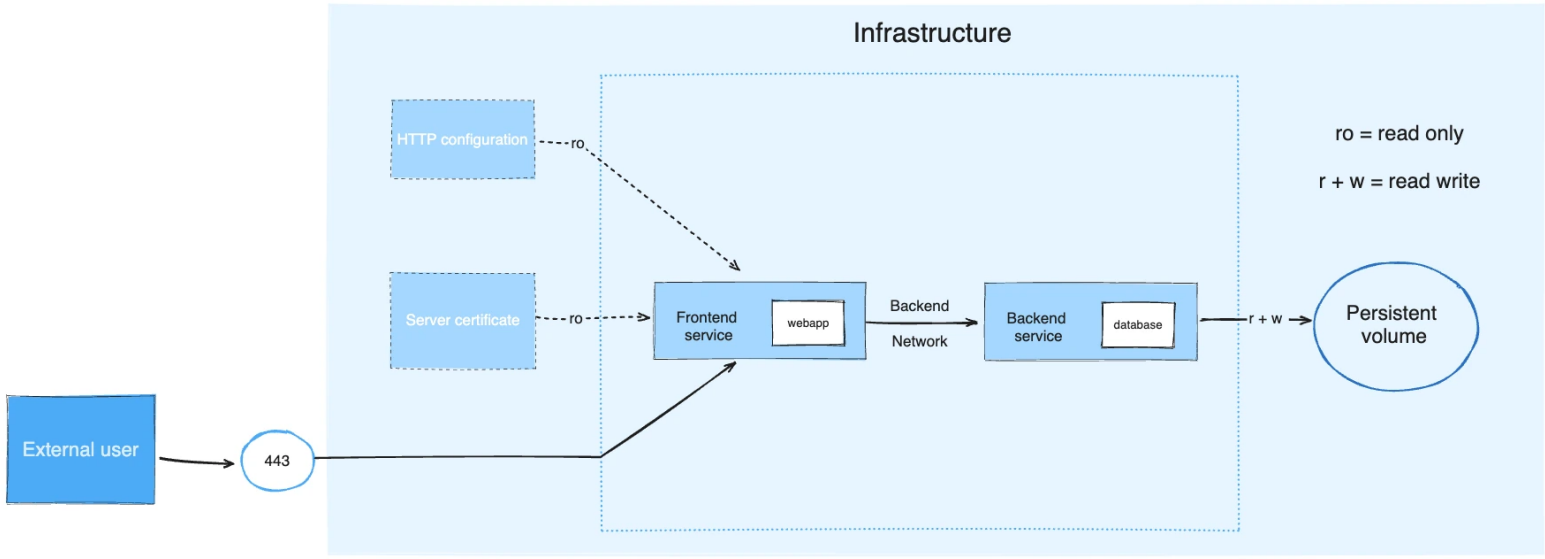
Начин рада

- `docker-compose.yaml` фајл с дефиницијом **сервиса**, **мрежа** и **волумена** долази у коренски директоријум апликације
- користимо `Docker CLI` за следеће основне команде:
  - `docker compose up` - покретање свих сервиса дефинисаних у `docker-compose.yaml`
  - `docker compose down` - заустављање и брисање свих сервиса дефинисаних у `docker-compose.yaml`
  - `docker compose ps` - излиставање свих сервиса са њиховим тренутним статусом
  - `docker compose logs` - приказивање излаза на појединим сервисима у циљу дебеговања

[24] (енг. multi-container applications)

Апликације са више контејнера[26]

Пример: Веб апликација пуног стека



Извор: `https://docs.docker.com/compose/intro/compose-application-model`, датум приступа: 2. октобар 2024.

[26] (енг. multi-container applications)