NERD

Úvod pro vývojáře

(ver. 2, 25. 6. 2019)

Václav Bartoš

Osnova

- Co je to NERD + ukázka
- Princip fungování, architektura, UpdateManager a moduly, konfigurace
- Uspořádání repozitáře
- Jak vytvořit vlastní modul
- Aktuální moduly, datový model
- Nová architektura
- Plány do budoucna

Co je to NERD?

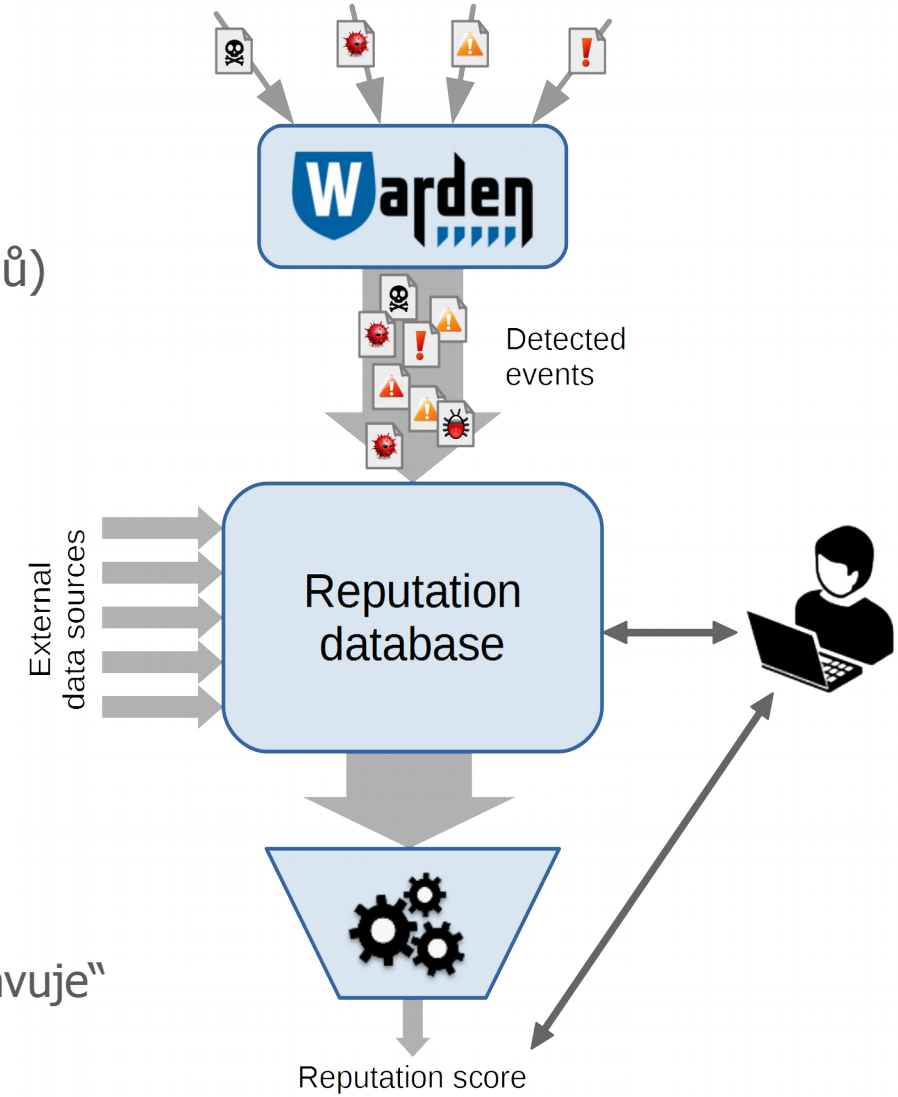
Co je to NERD

- Databáze síťových entit
  - **IP adresy**, sítě, domény, …
  - Seznam známých zdrojů škodlivých aktivit na internetu a všeho, co o nich víme
  - Mnohem víc, než jen blacklist

NERD
NETWORK ENTITY
REPUTATION DATABASE

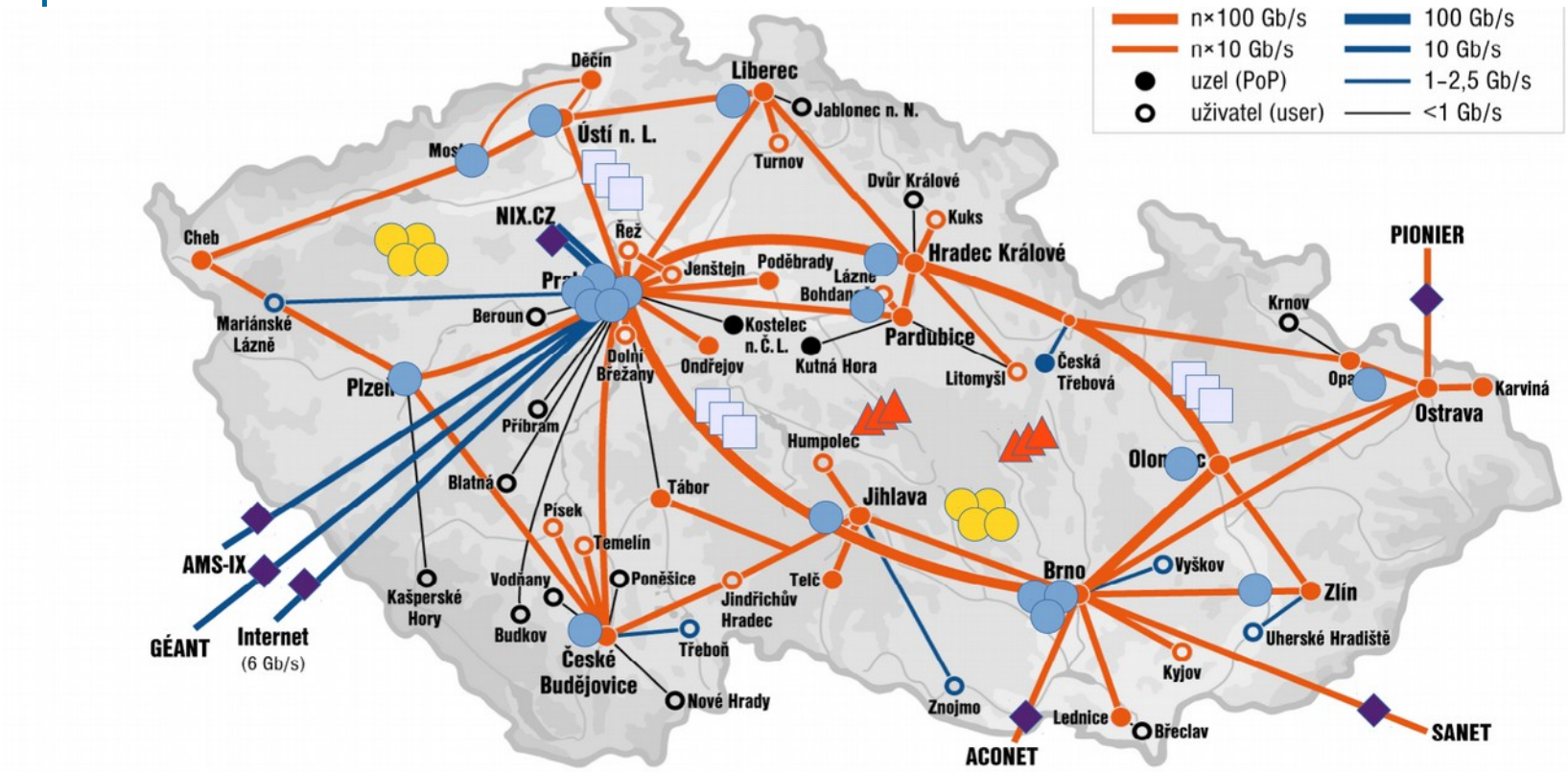
Co je to NERD

- Základní princip fungování:
  - Přijímá hlášení o bezpečnostních událostech z Wardenu (příp. i z jiných obdobných systémů)
    - Agregace podle zdrojové adresy
  - Obohacení o další data z externích zdrojů
    - Hostname, ASN, geolokace, …
    - Přítomnost na blacklistech
    - Open[DNS,NTP,…] resolvers
    - TOR exit nodes
    - …
  - Shrnutí všech informací do „reputation score"
    - Ohodnocení „jak velkou hrozbu entita představuje"



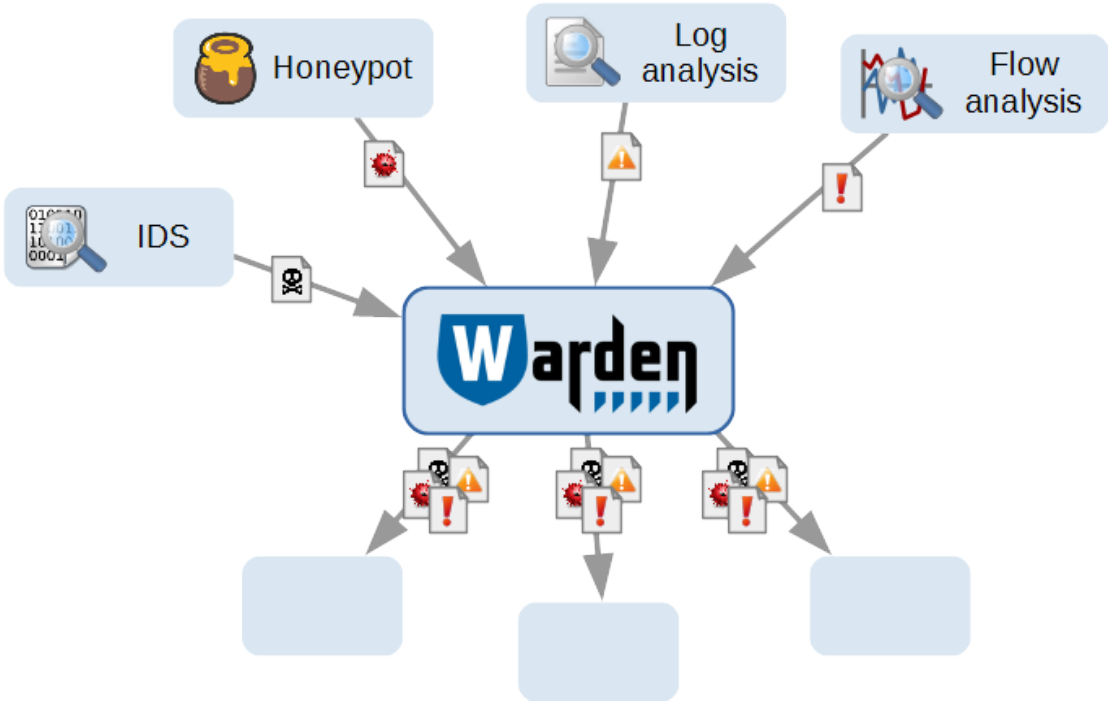
[Vsuvka - Warden]

- V síti CESNET a na univerzitách je množství systémů pro detekci škodlivého provozu



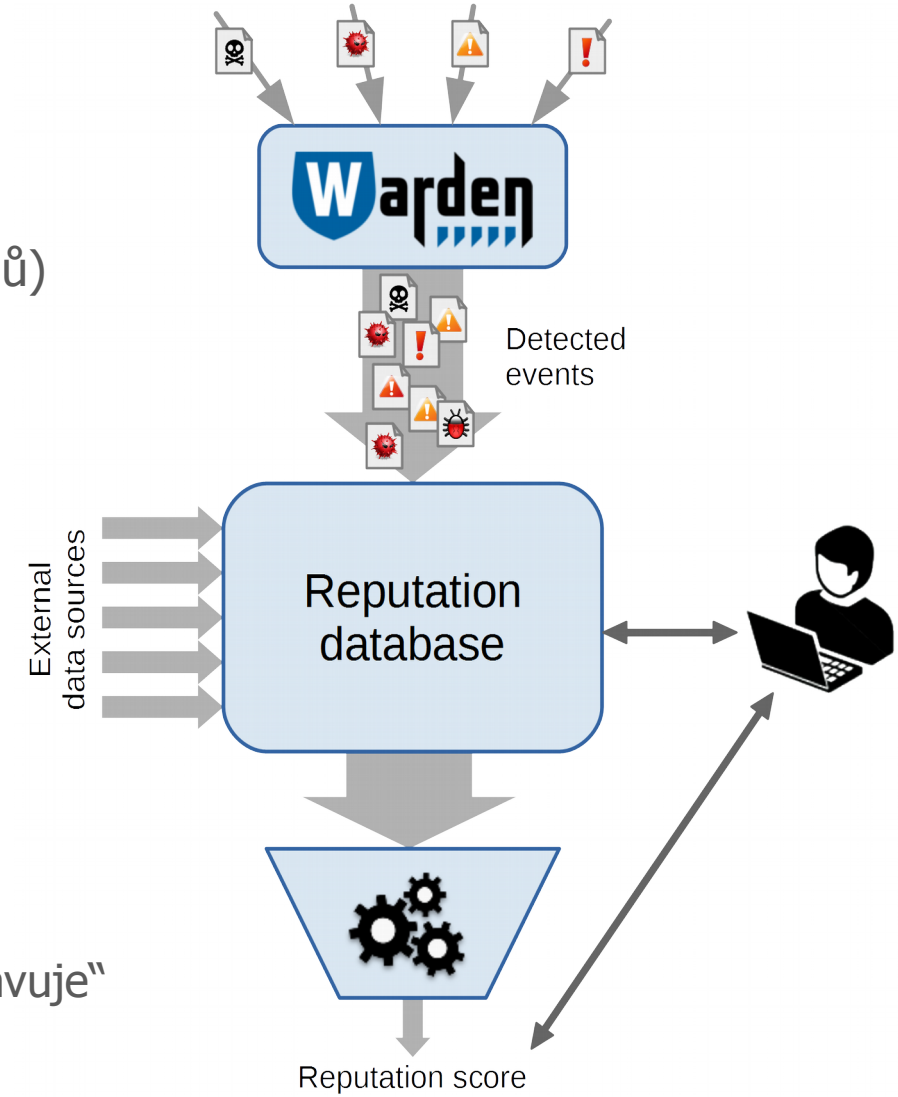
[Vsuvka - Warden]

- Warden – systém pro sdílení „alertů“ v CESNETu
  - Detektory posílají zprávy o detekovaných událostech na centrální server, ten je distribuuje přijímajícím klientům
  - Komunitní přístup: každý, kdo provozuje detektor a sdílí data, má přístup k datům všech ostatních
  - Jednotný datový formát (IDEA, https://idea.cesnet.cz/)



Co je to NERD

- Základní princip fungování:
  - Přijímá hlášení o bezpečnostních událostech z Wardenu (příp. i z jiných obdobných systémů)
    - Agregace podle zdrojové adresy
  - Obohacení o další data z externích zdrojů
    - Hostname, ASN, geolokace, …
    - Přítomnost na blacklistech
    - Open[DNS,NTP,…] resolvers
    - TOR exit nodes
    - …
  - Shrnutí všech informací do „reputation score"
    - Ohodnocení „jak velkou hrozbu entita představuje"

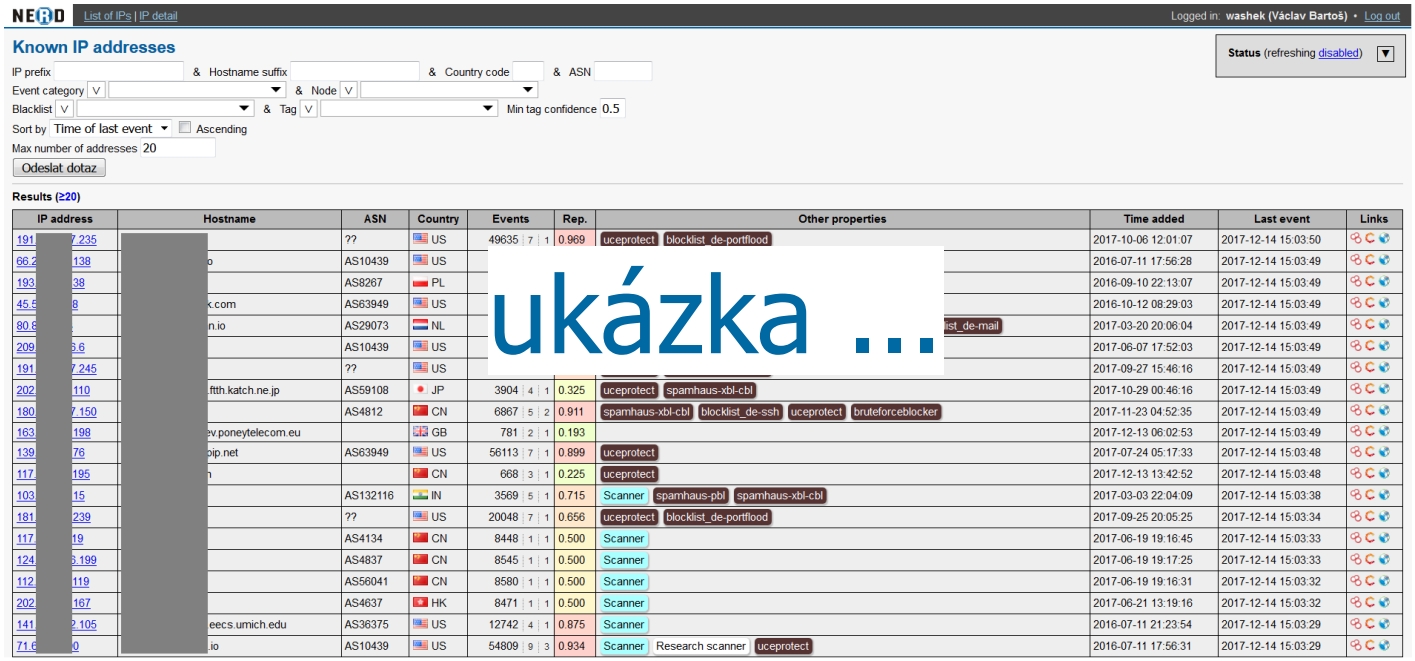


Co je to NERD

- Uživatelé
  - CSIRT týmy
  - Síťoví administrátoři
  - Výzkumníci v oblastí síťové bezpečnosti
  - (částečně i veřejnost)
- Možnosti použití
  - Upozornění na zlé adresy ve vlastní síti
  - Řešení a vyšetřování bezpečnostních incidentů
  - Generování blacklistů (blokování provozu z hodně škodlivých adres)
  - Statistiky, výzkum
  - Cokoliv dalšího, s čím někdo přijde :)
- Přístup k datům
  - Jednoduchý web + REST API

Odkazy

- Hlavní web:
  - https://nerd.cesnet.cz/
- Zdrojáky + wiki:
  - https://github.com/CESNET/NERD



Jak to celé funguje?

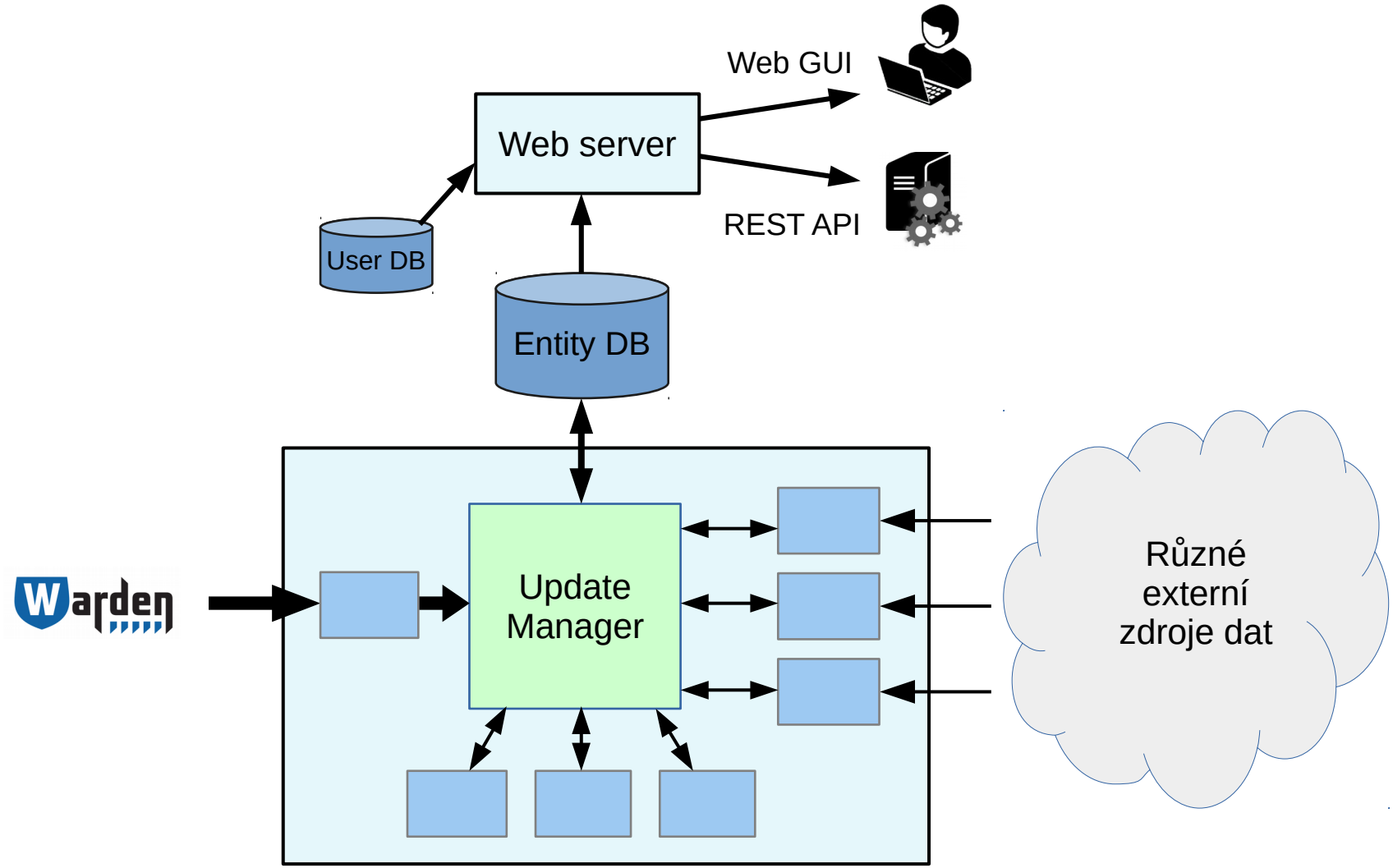
Princip fungování – záznamy entit

- Každá entita má v DB záznam
  - **IP adresa**, BGP prefix, ASN, IP block, organizace
    (pouze adresy nahlášené do Wardenu jako zdroj nějaké bezp. události)
  - Záznam = JSON dokument
- Záznam obsahuje
  - Metainformace o nahlášených událostech
    - počet událostí pro každý den, kategorii a detektor
  - Hostname, geolokace, přítomnost na blacklistech, různé tagy
  - Reputační skóre
  - Odkazy na BGP prefix, ASN atd., do kterých adresa patří
  - …
- Všechny informace jsou pravidelně aktualizovány
- Když k dané adrese nepřijde žádná událost po 14 dní, je záznam smazán

Princip fungování - moduly

- Systém je modulární – o každou část záznamu se stará konkrétní **modul** (plugin?)
  - Např. modul pro geolokaci, modul pro dotazy do blacklistů, atd.
- Všechny změny v záznamu řídí **UpdateManager**
  - Přijímá asynchronní požadavky na změny v záznamech
    - Např. při přijetí nové události z Wardenu
    - Pravidelné aktualizace záznamů
  - Volá funkce z jednotlivých modulů
  - … a zapisuje všechny změny do DB

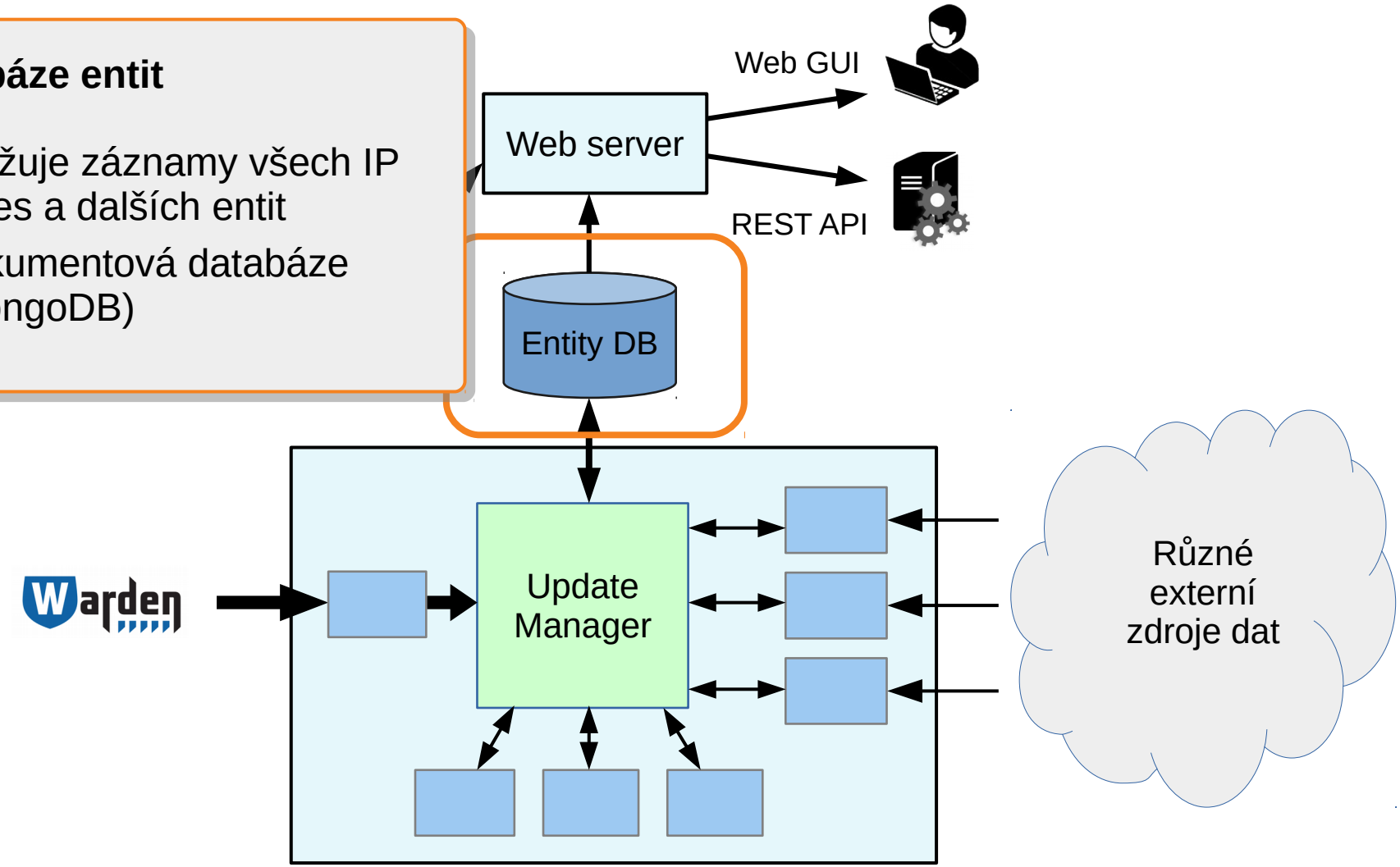
Logická architektura



Logická architektura

Databáze entit

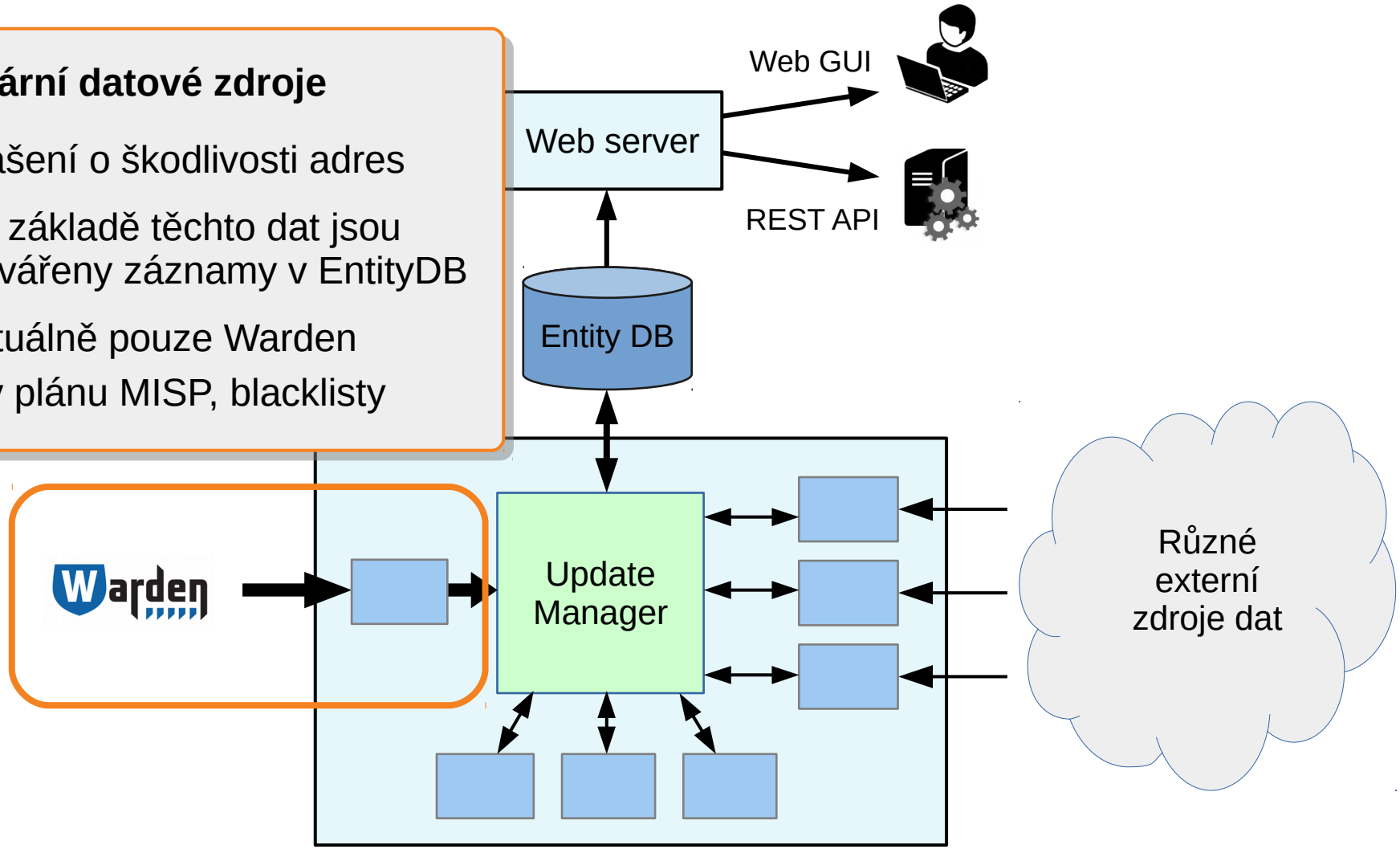
- Udržuje záznamy všech IP adres a dalších entit
- Dokumentová databáze (MongoDB)



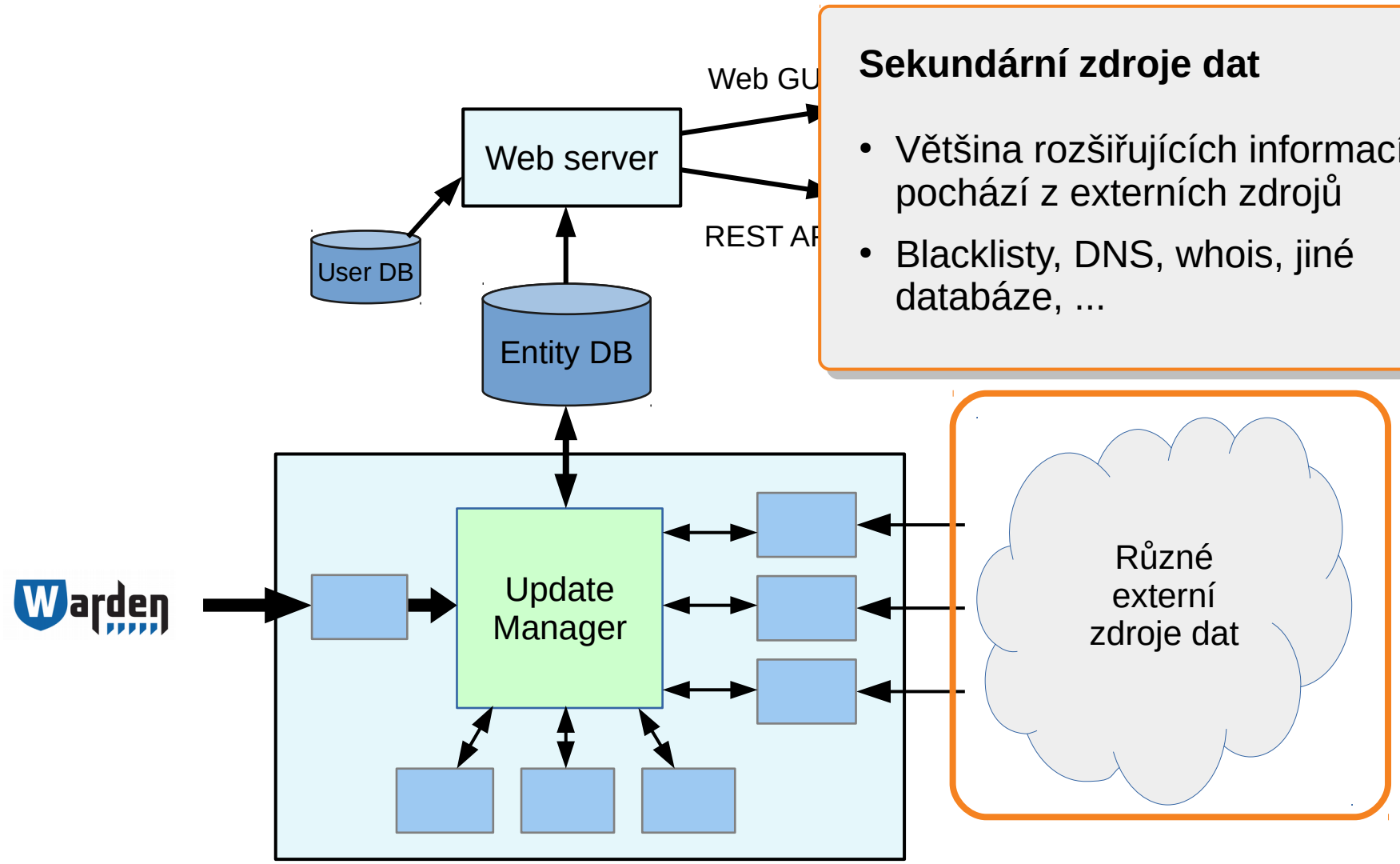
Logická architektura

Primární datové zdroje

- Hlášení o škodlivosti adres
- Na základě těchto dat jsou vytvářeny záznamy v EntityDB
- Aktuálně pouze Warden
  - v plánu MISP, blacklisty



Logická architektura



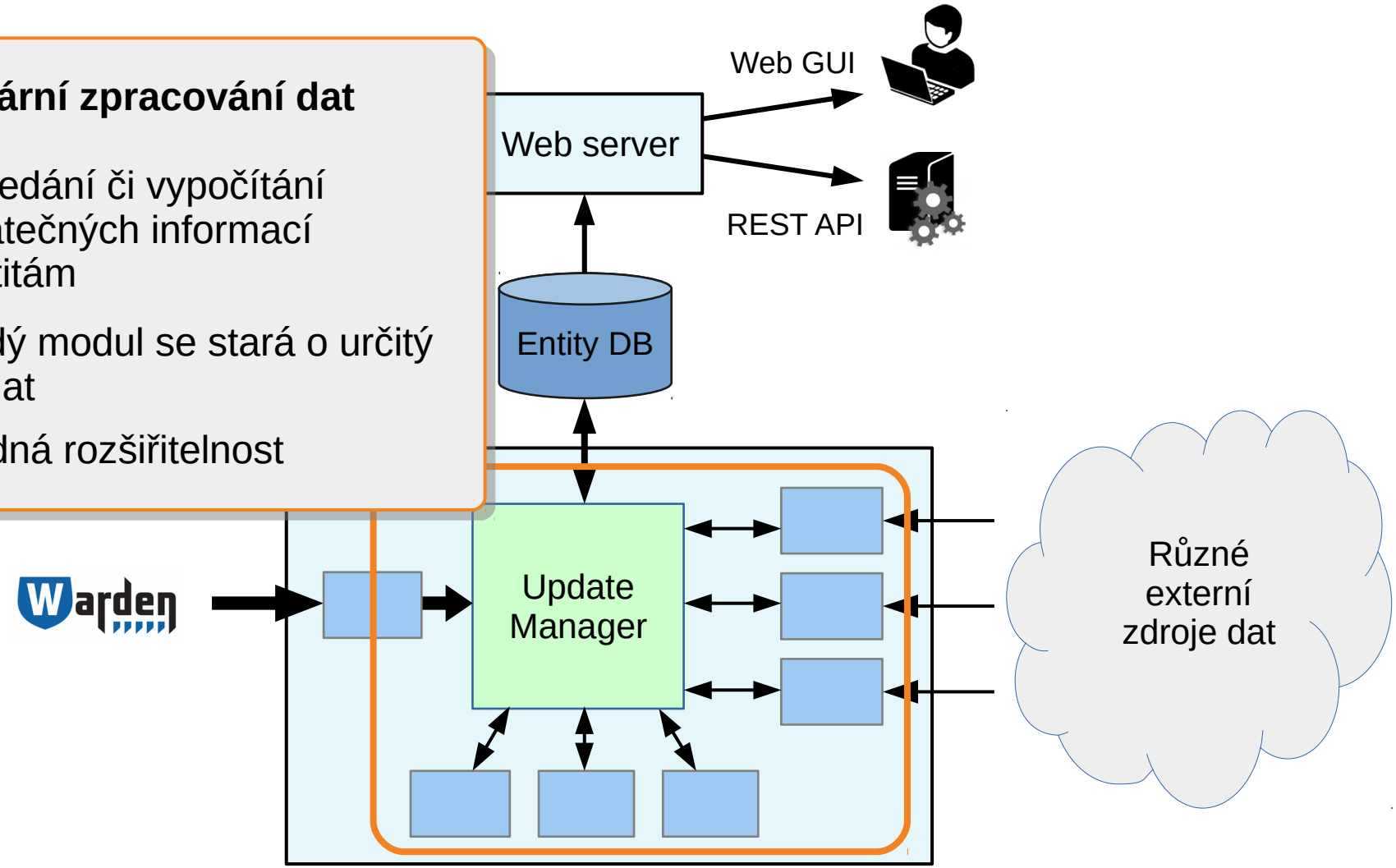
Sekundární zdroje dat

- Většina rozšiřujících informací pochází z externích zdrojů
- Blacklisty, DNS, whois, jiné databáze, ...

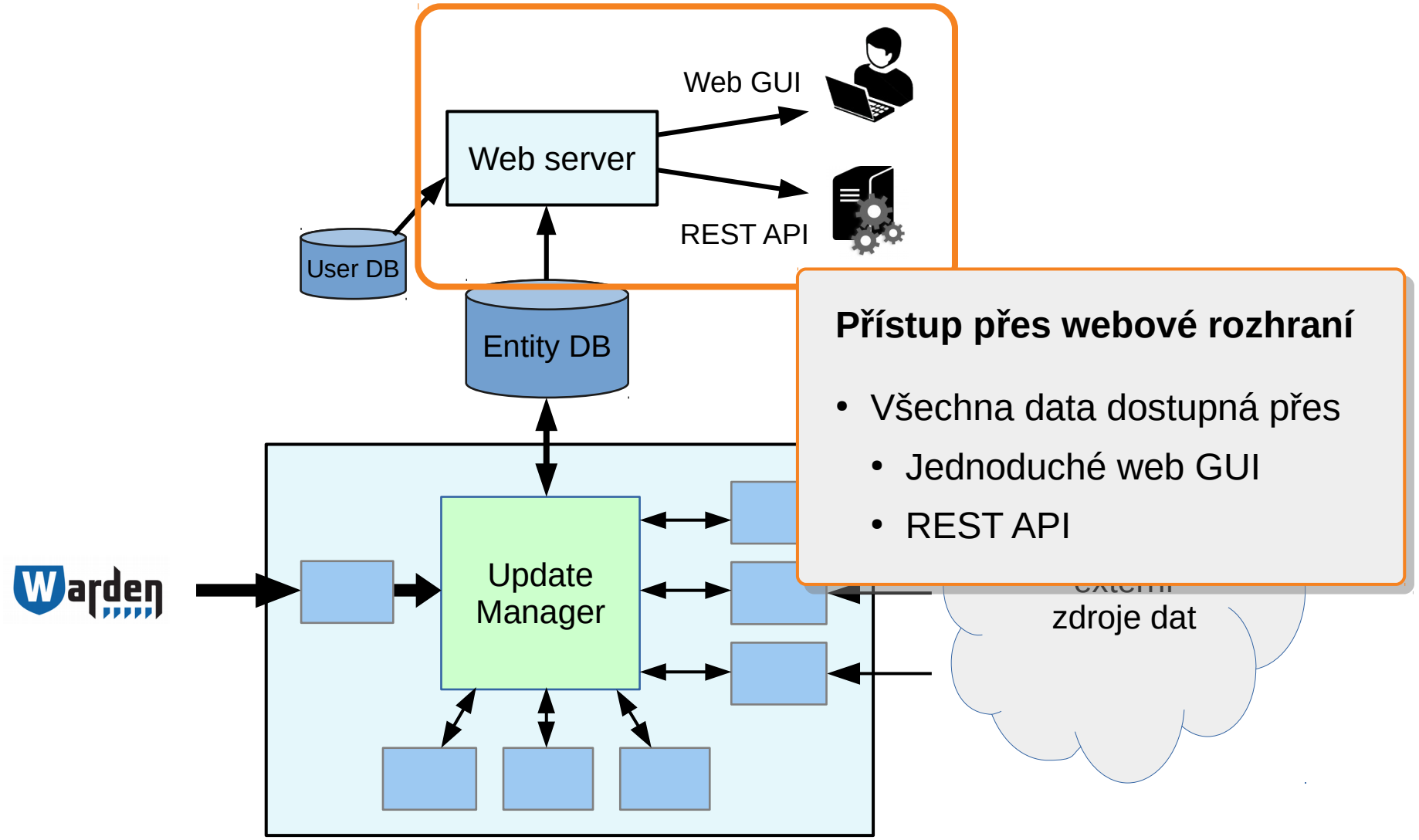
Logická architektura

Modulární zpracování dat

- Dohledání či vypočítání dodatečných informací k entitám
- Každý modul se stará o určitý typ dat
- Snadná rozšiřitelnost



Logická architektura



Přístup přes webové rozhraní

- Všechna data dostupná přes
  - Jednoduché web GUI
  - REST API

Několik pojmů

- **Entity key (ekey)**
  - Identifikátor záznamu, dvojice (entity_type, entity_id)
  - Např.: `("ip", "1.2.3.4"), ("asn", 2852)`
- **Atribut záznamu**
  - Název (klíč) položky v JSONu
  - Např.: `"rep", "geo.ctry"`
- **Update request / task**
  - Žádost o nějakou změnu v záznamu (či vyvolání pojmenované „události")
  - n-tuple (operace, klíč, hodnoty/parametry, ...) (parametr obvykle 1, může být i více nebo žádný)
  - Např.:

```
("set", "geo.ctry", "CZ")
("add", "events_meta.total", 1)
("event", "!every1d")                # pozn.: název události by měl vždy začínat "!"
```

  Seznam operací - viz začátek souboru NERDd/core/update_manager.py
  - Událost (event) přímo nic nemění
  - Speciální události
    - `!NEW` – generována automaticky UpdateManagerem při přidání nového záznamu
    - `!DELETE` – požadavek o vymazání celého záznamu (generováno modulem, UpdateManager pak provede výmaz)

Několik pojmů

- **Update request / task**
  - Žádost o nějakou změnu v záznamu (či vyvolání pojmenované „události")
    - Moduly nikdy needitují záznamy přímo
  - N-tuple:

    `(operace, klíč, hodnoty/parametry, …)` (parametr obvykle 1, může být i více nebo žádný)
  - Např.:

```
("set", "geo.ctry", "CZ")
("add", "events_meta.total", 1)
("event", "!every1d")                    # pozn.: název události by měl vždy začínat "!"
```

  - Událost (event) přímo nic nemění
    - Ale může na ni být zaregistrovaná callback funkce (viz dále)
  - Speciální události
    - `!NEW` – generována automaticky UpdateManagerem při přidání nového záznamu
    - `!DELETE` – požadavek o vymazání celého záznamu (generováno modulem, UpdateManager pak přeruší zpracování a záznam vymaže)

Seznam všech podporovaných operací - viz začátek souboru NERDd/core/update_manager.py

Několik pojmů

Handler function

- Callback funkce zaregistrovaná na změnu určitého atributu či událost
- Zpravidla definovány v modulech
- Prototyp:

```
def <name>(self, ekey, rec, updates):
```

- Vrací seznam dalších update requestů, které se mají na danou entitu aplikovat, nebo `None`
- Např. geolokační modul má funkci `geoloc` zaregistrovanou na událost `!NEW`, funkce zjistí lokační údaje dané IP adresy a vrátí 3 update requesty:

```
[
  ('set', 'geo.ctry', ctry),
  ('set', 'geo.city', city),
  ('set', 'geo.tz', tz)
]
```

Viz NERDd/modules/geo.py

Hlavní fronta update requestů

- Obsahuje záznamy ve formátu

```
(entity_key, [(op, key, value), (op, key, value), …])
```

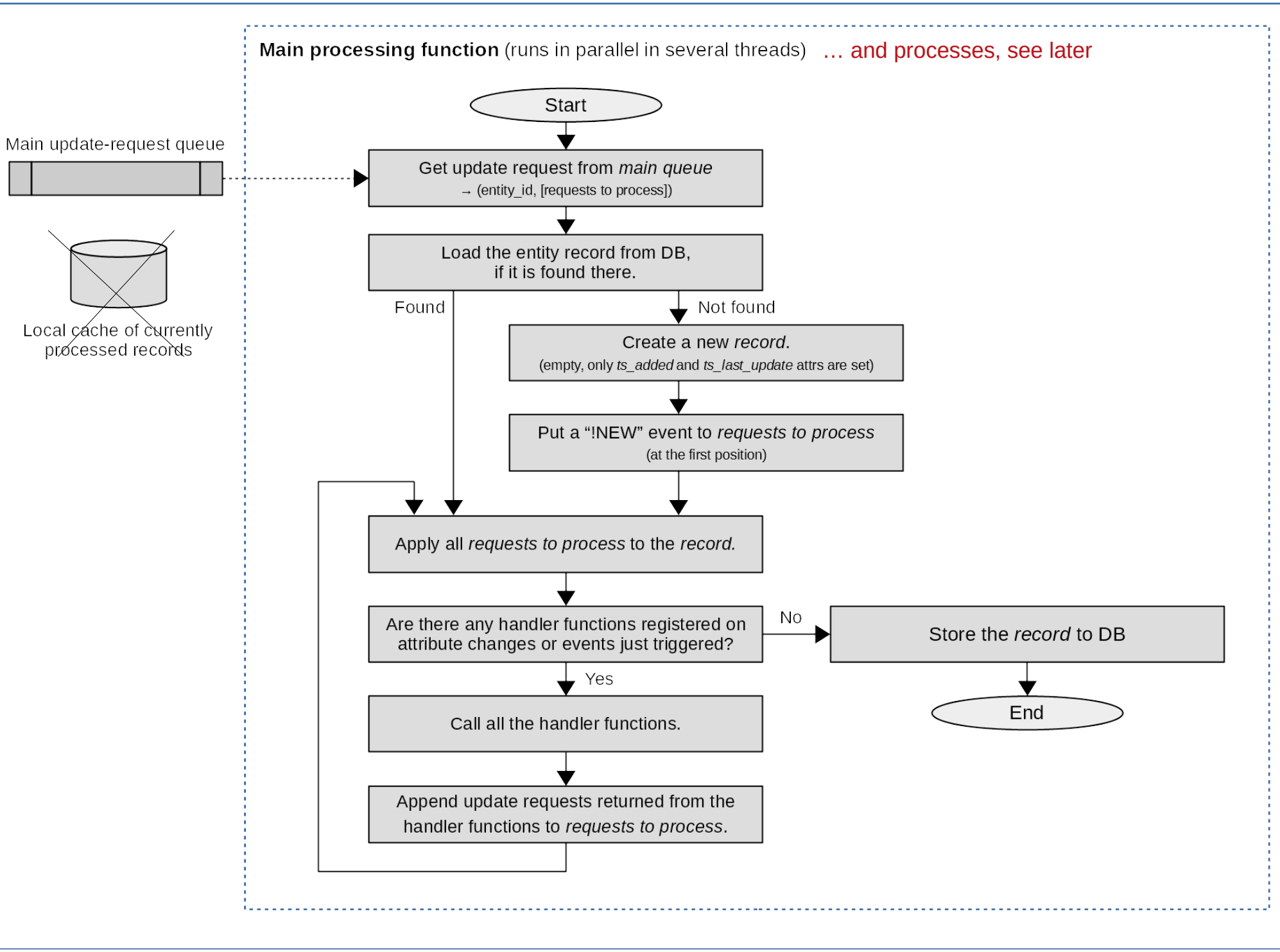
- Jakýkoliv modul do ní může asynchronně zapisovat

UpdateManager

1) Načte požadavek z hlavní fronty update requestů
2) Načte záznam dané entity (dle `entity_key`)
   - Pokud neexistuje, vytvoří nový záznam a na začátek seznamu update_requestů přidá událost !NEW
3) Provede požadované změny v záznamu
   - Pokud jsou na změněné atributy či události zaregistrovány nějaké handler funkce, zavolá je
   - Update requesty vrácené z handler funkcí přidá do seznamu requestů ke zpracování a opakuje 3), dokud nějaké requesty zbývají
4) Uloží změněný záznam do DB

(plus trochu „inteligence", aby se nevolaly handler funkce na atributy, které se ještě můžou změnit – v takovém případě se volání odloží)

Update manager



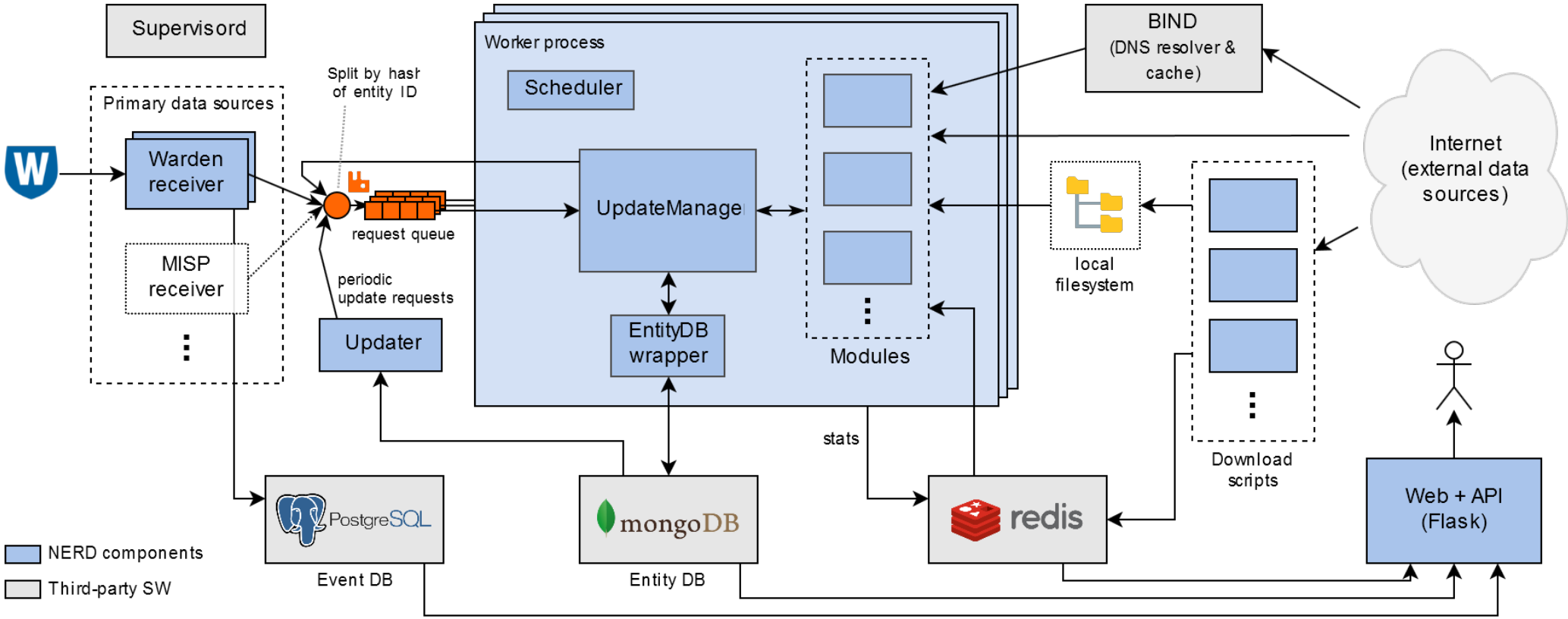
UpdateManager

- UpdateManager běží paralelně v mnoha vláknech
  - A spolu s ním tedy mohou běžet paralelně i handler funkce v modulech
  - V nové paralelní verzi navíc běží celá sada UpdateManager + moduly (= „worker“) ve více procesech
  - Aby nedocházelo k tomu, že jeden záznam bude zároveň upravován více instancemi UpdateManageru, ani se nemusely záznamy zamykat, **rozděluje se práce na jednotlivé procesy a vlákna podle hashe entity_key**. Každá entita je tak vždy zpracovávána jednou konkrétní instancí UpdateManageru a tedy sekvenčně.

- Další poznámky:
  - Pokud chce handler funkce provést změny v jiné než aktuální entitě, musí vložit nový update_request do hlavní fronty.
  - Každý modul si může vytvořit vlastní vlákno, které běží bez ohledu na handler funkce.

Architektura

Skutečná paralelní architektura



Fronta událostí

- Hlavní fronta update requestů / tasků („task queue“)
  - Sada front v RabbitMQ
  - Dvojice front pro každý proces
    - *normální* a *prioritní*
    - Normální má omezenou velikost a zápis do ní může blokovat
      - Použito všemi komponentami mimo worker
    - Prioritní nikdy neblokuje
      - Použito pouze pro tasky zadané handler funkcemi v rámci zpracování jiného tasku (bez prioritní fronty by takto mohl worker zalbokovat sám sebe)
  - V Pythonu wrapper (tváří se to jako jedna fronta)
    - TaskQueueReader
    - TaskQueueWriter
  - Každý proces si tasky dál interně dělí do vláken (četní a rozdělování je součást UpdateManageru, opět podle hashe)
  - V současnosti musí Writer vždy vědět, kolik je v systému worker procesů, sám spočítá hash a vloží do fronty příslušného workera (resp. nastaví *routing key*)

Pro management front existují skripty scripts/rmq_*.sh

Viz common/task_queue.py

Scheduler

- Umožňuje zaregistrovat pravidelné volání jakékoliv funkce
  - (wrapper nad Python modulem **apscheduler**)
  - Takový interní „cron“
  - Použito např. modulem **Updater**, který pravidelně pro každou entitu vydává události `!every1d` a `!every1w` (jednou za den/týden). Ostatní moduly mohou zaregistrovat své handler funkce na tyto události a tak provádět pravidelné updaty (díky tomu navíc všechny updaty jedné entity probíhají současně, což je výhodné z hlediska počtu přístupů do DB).

Zatím není jisté, jestli to v nové verzi vůbec bude potřeba.

Updater je tam samostatná komponenta mimo workery.

Ukládání raw IDEA událostí

- Kopie IDEA událostí z Wardenu může být ukládána v PostgreSQL
  - V současnosti se na ostrém serveru nepoužívá – čte se pomocí API z Mentatu
  - Ale stále je to podporováno (např. pro testování)
  - Zapnutí/vypnutí v konfiguraci
    - Klíč „eventdb“ v nerd.yml

Konfigurace

- NERD se konfiguruje pomocí souborů ve formátu YAML
  - Strukturovaná data jako např. JSON, ale syntaxe pohodlnější pro ruční psaní + podpora komentářů
- Tři hlavní soubory:
  - nerd.yml – konfigurace potřebná backendem i forntendem
  - nerdd.yml – konfigurace pouze pro backend
  - nerdweb.yml – konfigurace pouze pro frontend
  - (mohou existovat i další soubory, např. tags.yml s definicí pravidel pro přidělování tagů, do budoucna asi i další)
- V programu dostupné v `g.config` jako třída „hierarchical_dict"
  - Chová se jako normální Python `dict`, ale místo `config["whois"]["asn_file"]` lze napsat `config["whois.asn_file"]`
  - Příp. u volitelných parametrů s defaultní hodnotou `config.get("whois.asn_file", "/tmp/nerd-whois-asn.csv")`

Implementace v common/config.py

Uspořádání repozitáře

Uspořádání repozitáře

- https://github.com/CESNET/NERD/
- NERDd – (NERD daemon) zdrojáky backendu, tedy vše o čem jsme zatím mluvili
- NERDweb – zdrojáky webového rozhraní a API
- common – části společné pro backend a frontend
- scripts – různé pomocné skripty
- etc – konfigurační soubory
- install – instalační skripty, konfigurace supervisord a dalších služeb (Apache, DB, ...)
- doc – zdrojáky schémat z dokumentace

- Pozn: Dokumentace (značně nekompletní) je v github wiki: https://github.com/CESNET/NERD/wiki

/NERDd

- /NERDd/
  - core/ – základní komponenty
  - modules/ – moduly
  - worker.py – worker proces
  - updater.py – komponenta vydávající update_requesty s pravidelnými updaty
  - warden_receiver.py – vstupní (primární) modul pro čtení dat z wardenu
  - … - případně další přímo spustitelné komponenty
  - g.py – modul s globálními proměnnými
    - Jsou zde uložené reference na načtenou konfiguraci a instance základních komponent, např. `g.config`, `g.um`, `g.scheduler` (dostupní ve worker procesu)

/NERDweb

- /NERDweb/
  - static/ – obrázky, css, js
  - templates/ – Jinja2 šablony pro generování HTML
  - nerdweb.py – hlavní kód webu i API (zatím je téměř vše zde)
  - wsgi.py – wrapper pro WSGI (sem míří konfigurace web serveru)
  - userdb.py – vše kolem uživatelských účtů
  - ctrydata.py – statická data s názvy zemí

- Pozn.:
  - Web je implementován v Python knihovně Flask.
  - Bližší popis ale v této prezentaci není.

/common

- /common/
  - config.py – třída reprezentující načtenou konfiguraci
  - eventdb_psql.py – wrapper nad databází událostí (IDEA zpráv z Wardenu)
    - Na ostrém serveru se ale nepoužívá a čte se pomocí API z Mentatu
  - utils.py – různé drobné funkce (aktuálně jen parser času v RFC3339)
- /scripts/
  - TODO
- /install/
  - TODO (celou sekci o instalaci)

- Práce s gitem
  - Drobné a řádně otestované změny je možné dávat přímo do master
  - Pro větší změny vytváříme vlastní větve, které se pak do master mergují
  - Pokud se nedohodneme jinak, udělejte po dokončení práce pull request, já ho zkontroluju/okomentuju a když je vše pořádku, mergnu
  - Commit messages:
    - Začínají vždy „nazev_modulu:“, příp. „NERDd:“, „NERDweb:“, „scripts:“ apod.
    - Následuje stručný popis změn
    - Dobré příklady:
      - „Updater: fixed bugs preventing ASNs from updating“
      - „API: added support for bulk queries of IP address reputation (#17)“
    - Špatný příklad:
      - „Fixed exception when division by zero“

Aktuální sada modulů

Jaká data se ukládají

Základní moduly

- EventReceiver
  - Přijímá události s Wardenu
  - Ukládá kopie do EventDB (PostgreSQL)
  - Vydává update requesty pro úpravy položek v events a event_meta
  - Běží ve vlastním vlákně, nemá žádnou handler funkci
- Updater
  - Pravidelně vydává události `!every1d` a `!every1w` pro každou entitu
  - Rozloženo v čase
  - Do záznamů jsou přidány položky `_nru1d`, `_nru1w` (*next regular update*)
    - obsahují čas dalšího plánovaného updatu
    - Updater každých pár sekund vyčte z DB všechny entity s `_nru*` > aktuální čas a pro každou vydá příslušnou událost

Základní moduly

- Cleaner
  - Pročišťuje záznamy od starých informací
    - Meta informace o událostech starších než 90 dní
    - Podobně pro blacklisty
  - Nechává smazat celé staré záznamy

- Projít další moduly přímo v repozitáři …

- Projít datový model …
  - https://github.com/CESNET/NERD/wiki/Attributes

Vytvoření vlastního modulu

Vytvoření modulu

1) Nový soubor v /NERDd/modules/

- Obsahuje jednu třídu pojmenovanou podle funkce modulu (dědí z `BaseModule`)
- V `__init__` může zaregistrovat jednu nebo víc handler funkcí

  `g.um.register_handler(self.func_name, ent_type, triggers, changes)`

  (viz `register_handler` v update_manager.py)
- Může definovat funkce `start()` a `stop()`, ve kterých může např. vytvořit/zrušit nové vlákno

2) Přidání modulu ve worker.py

- Import modulu
- Přidání instance třídy do `module_list`

Logování

- NERD používá standardní Python knihovnu **logging**
  - Každý modul si v `__init__` vytvoří vlastní logger

    ```
    self.log = logging.getLogger("ModuleName")
    ```
  - Logy pak lze vypisovat takto:

    ```
    self.log.info("This is log message number {}.".format(i))
    ```
  - Výsledek:

```
2018-03-13T13:15:54,UMWorker-13,ModuleName,[INFO] This is log message number 5.
```

  - Funkce loggeru:
    - `.debug()`
    - `.info()`
    - `.warning()`
    - `.error()`
  - Defaultně se loguje od `INFO` výš
  - Override pro vlastní modul: `self.log.setLevel("DEBUG")`

Vytvoření modulu

- Další poznámky:
  - Docstring souboru i třídy by měl popisovat, co modul dělá.
  - Pokud modul potřebuje konfiguraci, stačí do nerd.yml nebo nerdd.yml přidat další záznam
    - V modulu bude dostupný v `g.config["asdf.qwer.xyz"]`
  - Handler funkce v modulu by měly pracovat jen s konkrétním záznamem
    - Můžou běžet paralelně ve více vláknech/procesech - pozor na jakákoliv sdílená data
    - Záznam entity se nikdy nemění přímo – pouze prostřednictvím vrácení seznamu update requestů
    - Pokud je třeba změnit (vytvořit) jinou entitu:
      `g.um.update(entity_key, list_of_update_requests)`
  - Není problém přidat závislost, pokud je to balíček v PyPI
    - Přidat do /NERDd/requirements.txt

cesnet

DĚKUJI ZA POZORNOST

MÁTE NĚJAKÉ DOTAZY?