Eredeti angol forrás: https://12factor.net/

1. Kódbázis

Code repository

Az ideális alkalmazás forráskód módosításait verzió követő rendszerben érdemes tartani és nyomon követni. (Subversion, Mercurial, Git) Ez még akkor is igaz, ha 'egy emberes' projektet fejlesztünk.

A kód tár (code repository) nem más, mint a módosítások adatbázisának egy 'saját' másolata. (rövidítve 'kód repó', szimplán 'repó')

A 'kódbázis' lehet bármilyen repó (centralizált verziókövető esetén Subversion) vagy a repó bármilyen részhalmaza, amelyik tartalmazza a 'root commit'-ot (a kezdeti commitot).

Alkalmazás vs. elosztott rendszer

Mindig 1-1 megfeleltetés legyen a kódbázis és az alkalmazás között:

- ha több kódbázisunk van, az soha nem lehet egy alkalmazás, hanem ilyenkor 'elosztott rendszer'-ről beszélünk.

Az elosztott rendszer minden komponensét külön alkalmazásnak kell tekinteni. Ha több alkalmazás ugyanazt a kódbázist használja (arra épül), az mindenképpen hibás megközelítés. Ha nem tudjuk elkerülni ezt az esetet, akkor több alkalmazás által közösen használt kódot olyan könyvtárakra (lib) kell felosztani, amit valamilyen függőségkezelő (dependency manager) kezel. Ez egyszerű refactoring módszerekkel könnyen elérhető.

Deployment

Egy alkalmazásnak minding egy kódbázisa van, de több telepítése, telepítési változata (deploy) lehet. A 'deploy' az alkalmazás egy éppen futó példánya. Ez általában valamilyen 'production site', éles nyilvánosságra hozott változat. Fontos feltétel, hogy minden fejlesztő képes legyen helyi futtató környezetet kialakítani (local development environment) - (ami lehet valamilyen felhőrendszerben is) - amelyek 'deploy'-nak, telepítésnek tekinthetőek.

Developer -> Staging -> Production

A kódbázis nem változik a telepítések között, annak különböző verziói lehetnek aktívak telepítésenként. (pl. tesztelési célból) Egy fejlesztőnek sok olyan commit-ja lehet, amelyet még nem zárt le (nincs staging fázisban). A staging fázisban lévő commitok még nincsenek deploy-olva, és nem tekinthetőek production változatnak. De ettől függetlenül ugyanazt a kódbázist használják és egymástól egyedi azonosítóval megkülönböztethetőek.

Commitok megjegyzései mindig rövid tömör mondatok legyenek. (pl. utalva a bug tracker rendszerbeli azonosítóra)

2. Függőségek

Packaging system

A legtöbb programozási nyelv rendelkezik valamilyen központi csomagkezelő rendszerrel. (perl → CPAN, Rubygems → Ruby, npm - nodejs, Python - pip, c/c++ - autoconf). A könyvtárak (előre definiált verziója) a csomagkezelő rendszer segítségével telepíthetőek, operációs rendszer szinten vagy alkalmazás szinten (virtual environment → python), amikor a függőség 'scope'-ja a telepítési könyvtártól lefelé érvényes.

Dependency declaration manifest (függőség deklarációs jegyzék)

Minden függőséget és azok verzióit tárolja. Izolálja a developer és production módban használt függőségeket is.

A csomagkezelő rendszer alkalmazása egyszerűsíti az alkalmazás telepítését és új fejlesztő bevonását a projektbe.

3. Konfiguráció

Az alkalmazás konfigurációja *olyan egyedi beállítások halmaza, amelyik változhat a telepítések között*.

Az alábbi tipikus konfigurációs beállítások lehetnek:

- eszközkezelők adatbázisokhoz, memcached, mysql séma, kapcsolat, stb.
- elérési tokenek (API keys) - kódok külső szolgáltatásokhoz, pl. Amazon S3 felhőszolgáltatás, Facebook API
- telepítési konstansok - hostnevek pl. teszt/éles rendszerhez
- különböző logolási szinteket állíthatunk be

Az alkalmazás soha nem tartalmazhat konfigurációs paramétereket a forráskódban. Kivétel: belső alkalmazás konfigurációk, pl. 'routing' információ maradhat a kódban, mert az belső konfigurációnak számít.

Az alkalmazás konfigurációt érdemes környezeti változókban tárolni.

Mivel:

- a környezeti változókat könnyű megváltoztatni az egyes telepítések között, a kód módosítása nélkül
- nincs veszélye olyan véletlen commitnak, amivel azonosító kódok, jelszavak kerülhetnek a repóba.

Java-ban a környezeti változókat a System.getProperty() hívással elérhetjük, érdemes a kódban dinamikusan is alkalmazni.

4. Szolgáltatások

Ezek hálózaton elérhető olyan (sokszor külső) szolgáltatások, amelyek a normál működéshez elengedhetetlenek (MySql, MongoDB, ActiveMQ, RabbitMQ, SMTP, Amazon S3, Google Maps stb.) Rendszer adminisztrátorok kezelik hagyományos módon vagy külső szolgáltatók (third-party)

A rugalmas használatukhoz elkerülhetetlen, hogy:

- az alkalmazáskód nem különböztetheti meg a lokális és third party interfészeket
- a deploy-olt alkalmazásnak könnyen kell váltania a szolgáltatások között, kódmódosítás nélkül
- minden szolgáltatást erőforrásnak kell tekinteni. 1 MySQL adatbázis 1 erőforrás, 2 MySQL 2 különálló erőforrás
- attach és detach mechanizmusokat kell kialakítani, ha bármilyen hiba történik, akkor könnyen át lehessen kapcsolni a működőre szolgáltatásra

5. Build, Release, Run

A kódbázis általában három fázisban alakul át telepített változatra:

- **build stage**: olyan transzformáció, amely a kód repót futtatható kóddá alakítja. Kiválasztja a megfelelő függőségeket és kompilálja a bináris kódot és az asseteket (képek, css, js).
- **release stage**: a build stage-által generált build-et kombinálja a hozzá tartozó konfiguráció felhasználásával → release
- **run stage**: alkalmazás futtatása a futtató környezetben.

Következmény: a kód a release fázisban már nem módosítható, tilos bármilyen kézi módosítás.

A deployement tool-ok egyik fontos tulajdonsága, hogy vissza tudnak lépni egy korábbi release változatra. pl. Capistrano eszköz a release-eket külön könyvtárban tárolja és szimbolikus linken hivatkozik rájuk.

Általában minden release egyedi azonosítót kap. pl. release_2017_04_04_21:00:14, vagy egy növekvő azonosítót: v0121. Minden módosítás külön release-t kaphat!

A verzió követő rendszerekben sokszor 'TAG'-el jelölik ezeket a fontos commitokat.

6. Futó folyamatok

Úgy kell futtatni az alkalmazást, mint egy vagy több állapotmentes folyamatot.

A folyamatok között és azokon belül nincs és nem lehet adatmegosztás és adattárolás. Minden olyan adatot amelyet perzisztens módon kell tárolni, 'stateful backing service' fogja tárolni. Ha nem így használjuk a folyamatokat, akkor a későbbi skálázást nagyon megnehezíti.

Egy folyamathot rendelt memóriáját vagy fájlrendszer elérést 'tranzakció cache'-ként kell értelmezni.

Pl. egy nagyméretű fájl letöltésénél a fájlrendszert használjuk, feldolgozzuk a fájlt, majd az eredményt adatbázisban kell tárolni.

Nem feltételezhetjük, hogy bármi is cache-elve van a memóriában vagy a lemezen egy további

(HTTP) kéréshez. Még inkább azt sem, hogy majd más futó folyamatok is elérhetik ezt a lokális folyamatszintű cache-t.

Egy komponens újraindításakor azt kell feltételezni, hogy a lokális cache törlődni fog (törlődnie is kell!).

Vannak olyan rendszerek, pl. django compressor, amelyek a lefordított asseteket lokális cache-ben tárolják, ezek az assetek a compiling fázisban készülnek el és nem változhatnak futtatáskor.

Minden session adatot webes alkalmazások esetén kötelezően külső Memcached vagy az újabb Redis szolgáltatással érdemes tárolni, hogy a következő kérés egy másik folyamatból is el tudja érni.

7. Port binding

Webes alkalmazások általában egy webserver containerben futnak. pl. PHP alkalmazások Apache Httpd modulként futnak, vagy a Java alkalmazások Tomcat Servlet Containeren belül.

A webalkalmazás a HTTP elérését szolgáltatásként exportálja egy porthoz kötve. Lokális fejlesztésként a fejlesztő service URL-ként a http://localhost:6120 -címen éri el az alkalmazás szolgáltatását. Telepítéskor a rendszer a hostnevet (localhost) automatikusan kicseréli egy publikusra.

A port binding segítségével egy alkalmazás átalakulhat backing service-nek egy másik számára az URL segítségével.

8. Konkurencia

Minden futó alkalmazás a számítógépen egy vagy több folyamatként is reprezentálható. A webes alkalmazások különbözőképpen kezelik a futó folyamatokat. Egy PHP alkalmazás az apache web szerver gyermek folyamataként fut, annyi fut belőle amennyit a beállítások, valamint a dinamikus terhelés meghatároz.

A Java virtuális gép ezzel szemben a párhuzamosságot szálakkal oldja meg.

Mindkét esetben a fejlesztő minimálisan befolyásolja/látja a konkurenciát és annak megvalósítási különbségeit.

A web alkalmazások esetén meg kell különböztetni a **web** és **worker** típusú futó folyamatot.

- **Web process** - kiszolgál egy HTTP kérést.
- **Worker process** - egy hosszabban futó háttér taszk-ot hajt végre. (bármilyen asszinkron feladat, pl. szavazás, videó vágás, kép átméretezés)

Ez a megkülönböztetés nem befolyásolja, hogy egy VM ami egyébként egy worker process-t futtat, azaz valójában hány szálat használ.

Skálázás esetén pontosan lehet definiálni, melyik folyamattípusból mennyi fusson egy adott terhelési szinten. (természetesen az egyes típusok darabszáma nem lineáris a terhelés függvényében)

Soha nem szabad deamon-ként futtattni egy alkalmazást, vagy PID fájlokat írni. Mindig rá kell

hagyatkozni az operációs rendszer beépített folyamat menedzser eszközeire. pl. Foreman vagy Upstart. Ezek az eszközök gondoskodnak az általuk menedzselt folyamatok életciklusiról. Kezelik a felhasználói leállításokat és újraindításokat is.

9. Eldobhatóság

A folyamatokat úgy kell tervezni hogy könnyen eldobhatóak legyenek. A folyamatok indulási idejét minimalizálni kell. Gyors indítással a rugalmasság és a robusztusság is egyaránt nő. A folyamat menedzser könnyebben mozgathatja a folyamatokat akár egy új fizikai gépre is.

Graceful shutdown (kecses leállás) SIGTERM szignál esetén a webes folyamat megengedi a éppen aktuális kérés feldolgozását. Több kérést viszont nem enged végrehajtani. Ezzel a rendszer inkonzisztens állapotba esését megakadályozza.

A worker folyamat SIGTERM esetén visszahelyezi a job-ot az üzenetsorra.

'Hirtelen halál' esetén (sudden death) leáll a folyamat, pl. valamilyen hardver hiba lép fel vagy a futtató környezetben fellépő bármilyen hiba miatt, a tanácsolt megoldás olyan robusztus üzenetsor használatát írja elő, ami lecsatlakozás esetén visszavonja az utolsó job-ot a klienstől. Ezzel az újraindítás után nem lesz adatvesztés, sőt a felhasználó nem értesül/nem érzékeli a hibát.

10. Development/production hasonlóság fenntartása

Történetileg kialakult egy szakadék a fejlesztés és a végfelhasználás/éles üzem (deployment) között.

- időszakadék: a fejlesztő hetekig dolgozik a kódon, ameddig az végfelhasználásra kerül.
- személyi szakadék: a fejlesztő írja a kódot, az operátorok telepítik és indítják egy elkülönülő éles környezetben
- eszköz szakadék: a fejlesztő apache-ot használ Windows-on, a production változat pedig Nginx-et Linuxon

Ezért folyamatos integrációs 'Continuous Integration (CI)' módszereket kell alkalmazni

- időőszakadék csökkentése: kis kódrészletek után gyors kipróbálás
- személyi szakadék csökkentése: a fejlesztő maga készíti CI szkripteket és indítja a tesztrendszeren
- eszköz szakadék csökkentése: használjuk ugyanazokat az eszközöket, ha lehetséges

(Vagrant, Ansible, Docker)

11. Logok kezelése

A logokra úgy tekintsünk, mint esemény folyamokra.

A logok mutatják meg a futó alkalmazásaink viselkedését. Ezek szerver alapú környezetekben 'log fájlokban' tárolódnak. A logok valójában egy időrendben megjelenő kimeneti stream-ek amelyeket a futó és a háttérfolyamatok generálnak. A logok nyers szövegeket tartalmaznak, általában egy sor jelent egy eseményt. (a kivételek sokszor a verem információk miatt több sorosak is lehetnek).

Logoknak nincsenek jól definiált kezdetük és végük.

Fejlesztéskor a log üzeneteket érdemes a standard outputra írni, hogy közvetlenül lássuk az esetleges hibákat. A production környezetben a futtató környezet összegyűjti a logokat egy központi tárolóba, archiválás céljából.

- Log routereket érdemes használni: Logplex, Fluent
- vannak indexelt log feldolgozók is (Splunk, Hadoop/Hive)

Előnyök:

- specifikus események keresése a múltban
- trendek grafikus ábrázolása
- alertek definiálása felhasználói heurisztikák alapján (alert - ha a hibák mennyisége percenként elér egy határértéket)

12. Admin folyamatok

Az admin folyamatok egyszeriek

- adatbázis migráció (a modell változásai miatti adatbázis változások automatikus követése)
- futtatási konzol (a futó folyamatban ezzel kézi konfigurációs módosításokat lehet végezni)
- egyszeri szkriptek commitálás a repoba - deploykor az admin folyamatok automatikusan elfutnak

Admin folyamatok forráskódját commitálni kell a repositoryba, általában egy speciális helyre. Sokszor elnevezésük szabályhoz kötött: pl. 0012_db_migration.sql