Michał Bultrowicz

Curriculum Vitae

Kontakt

michal.bultrowicz@gmail.com
(+48) 790 467 660
https://bultrowicz.com/
https://github.com/butla

Podsumowanie

Jestem entuzjastycznym programistą przyzwyczajonym do tworzenia aplikacji kluczowych dla biznesu oraz do radzenia sobie z wyzwaniami i presją z tego wynikającymi. Kocham Pythona, Linuxa i każde oprogramowanie, które jest za razem eleganckie i solidne. Mocno wierzę we wprowadzanie zmian stopniowo i ciągłe testowanie (continuous testing). Lubię być na bieżąco z nowinkami w moich dziedzinach, ale zauważyłem, że inspiracja może przyjść właściwie skądkolwiek.

Potrafię projektować, implementować oraz konsultować tworzenie rozwiązań informatycznych, w szczególności:

- automatyzacje i pomiary procesów biznesowych („dashboardy”, statystyki, alarmy)
- „data pipelines”
- wysoce dostępne (highly available) aplikacje sieciowe („webowe”)
- zautomatyzowane testy (back-endowe, front-endowe, obciążeniowe, chaos)
- oprogramowanie oparte o Pythona

Mogę także przeszkolić Państwa kadrę IT w utrzymywaniu i dalszym rozwoju systemów które tworzę, abyście nie byli ode mnie zależni na zawsze.

Umiejętności komputerowe

- ekspert w programowaniu w Pythonie
- drugorzędne języki programowania: Bash, JavaScript, Java, C++, C#
- pisanie zrozumiałego i testowalnego kodu
- tworzenie aplikacji sieciowych z użyciem frameworków AioHTTP, Falcon, Flask, oraz Django
- zaawansowane testowanie aplikacji sieciowych (z mikroserwisami włącznie)
- zapewnianie wysokiej dostępności (HA)
- dobra wiedza na temat optymalizacji SQL dla PostgreSQL (i skalowanie go z użyciem CitusDB)
- praca z różnymi magazynami danych: Redis, Kafka (oraz AWS Kinesis), ElasticSearch, S3/MinIO, Amazon Redshift
- używanie kontenerów do „deploymentów” oraz testowania (Docker, Kubernetes, Helm)

- implementacja bezpieczeństwa w sieci (X.509, mTLS, JWT, OAuth2)
- zautomatyzowanie zarządzanie infrastrukturą z użyciem Terraforma i Ansibla
- inżynieria danych (budowanie „data pipelines” oraz systemów ETL)
- praca z chmurami AWS, GCP i DigitalOcean
- implementacja „Continuous Delivery” z pomocą Gitlab CD (Jenkinsem w przeszłości)
- trochę doświadczenia w budowanie front-endów z użyciem ReactJS
- płynność w używaniu GITa
- zdolny pracować jedynie w terminalu, bez środowiska graficznego

---

Umiejętności „miękkie”

- koordynowanie pracy w zespole Scrumowym lub Kanbanowym
- uczenie programistów przy pomocy dokładnych przeglądów kodu (code review)
- wiedza na temat zagadnień licencji oprogramowania i ich zgodności
- analiza napięć między zespołami w projekcie (czy zachęty są odpowiednio ustawione?)

Doświadczenie

2018.11 – 2021.01

Senior DevOps Engineer w British Telecom (BT)
Zdalnie, ze sporadycznymi spotkaniami w Londynie, Ipswich i Brentwood

Główny projekt: Vena (https://www.mediaandbroadcast.bt.com/vena.html)

Pracowałem w zespole zewnętrznych kontraktorów. Naszym zadaniem było stworzenie aplikacji sieciowej służącej do projektowania infrastruktury telekomunikacyjnej oraz do wstępnej konfiguracji routerów Juniper i Cisco. Żeby to osiągnąć musieliśmy intensywnie współpracować z inżynierami BT z różnych zespołów, zbierając informacje o aktualnych procesach firmy i pomagając projektować zamówione rozwiązanie, które miało się wpasować w większy projekt.

Mieliśmy być programistami uniwersalnymi, piszącymi kod, przygotowującymi jego „deploymenty” i działającymi ze sprzętem. Nie mieliśmy tradycyjnego szefa zespołu, co było interesującym ćwiczeniem w samoorganizacji. Mój zespół składał się z 4 do 8 osób (rósł), ale cały projekt składał się z wielu zespołów.

Ten projekt był także próbą przejścia na zwinne metodyki wytwarzania dla organizacji. Nasz zespół kontraktorów temu przewodził.

Ważniejsze wykorzystane technologie:

- Django i Django REST Framework – większość naszego back-endu i front-endu
- PostgreSQL – nasza główna baza danych
- Kubernetes i Helm – „deploymenty”
- Nginx – „reverse proxy”

- Docker Compose – uruchamianie lokalnej instancji aplikacji z jej zależnościami, do rozwijania kodu
- Minio – serwowanie statycznych plików i mediów naszej aplikacji
- Celery – asynchronicznie uruchamiane grafy zadań (wykorzystujące funkcjonalność "canvas")
- ReactJS – kilka bardziej dynamicznych stron front-endu
- Faust – czytanie strumieni wydarzeń z innych systemów z Kafki
- Redis – cache'owanie, broker Celery
- Sentry – śledzenie błędów
- Prometheus – metryki
- Junos PyEZ – zarządzanie routerami Juniper
- netmiko – zarządzanie routerami Cisco
- GitLab – repozytorium kodu / system „continuous delivery"
- Factory Boy – łatwe tworzenie obiektów ORM dla testów
- „terminal servers" – logowanie się przez SSH i uzyskiwanie terminala po porcie szeregowym do routerów

Moje nisze w zespole:

- projektowanie kodu i infrastruktury pod nowe funkcjonalności
- ustawianie automatycznych testów i przyspieszanie ich
- „refactoring" kodu pod testowalność
- dokładne przeglądy kodu (code review)
- uczenie innych członków zespołu zaawansowanego Pythona oraz testowalnego designu
- działanie na rzecz wybierania prostych i sprawdzonych open-source'owych rozwiązań które mogliśmy kontrolować ponad cięższe rozwiązania typu „enterprise", co do których zespół nie miał tyle ekspertyzy
- dbanie o „deploymenty" i infrastrukturę (razem z jednym innym członkiem zespołu)

2017.05 – 2018.09

Python Back-end Developer w Iterio Data (https://iteriodata.com/)
Zdalnie

Moim głównym projektem był system do analiz danych z kampanii reklamowych i ruchu po stronach internetowych. Pracowałem jako główny programista i faktyczny architekt systemu. Później przejąłem także utrzymanie i projektowanie naszej infrastruktury w chmurze. Poza tym robiłem trochę ETLa, sporadyczne proste analizy danych, oraz odświeżałem trochę starych skryptów automatyzacyjnych.

Ważna rzecz której się tutaj nauczyłem to to, że z odpowiednimi narzędziami i odpowiednimi inżynierami mały zespół (4 osoby, w tym przypadku) może stworzyć aplikację sieciową przetwarzającą dane milionów ludzi.

Moje główne obowiązki:

- tworzenie API webowych przy użyciu asynchronicznego Pythona (AioHTTP) i umieszczanie ich w kontenerach Dockerowych
- projektowanie naszych systemów na AWSie (ECS, Lambda, Kinesis, S3, CloudWatch)

- kodowanie naszej infrastruktury i przeprowadzanie deploymentów przy pomocy Terraforma
- tworzenie strategii i narzędzi testowych dla naszych aplikacji w chmurze
- pomoc w projektowaniu struktury naszej bazy danych w CitusDB (rozproszony PostgreSQL)
- ustawianie logowania (CloudWatch, Google Stackdriver), metryk oraz alarmów (Sentry, PagerDuty)
- wypracowanie naszych praktyk wytwarzania oprogramowania („trunk-based development”, „zero down-time deployments”, etc.)
- utrzymywanie starych skryptów, które używały korporacyjnych SOAPowych usług sieciowych (wprowadziłem Zeepa aby to ułatwić)

Pracowaliśmy w trybie typowym dla start-upów, a więc było dużo przełączania kontekstów i zawsze brakowało nam czasu. Przez to kod który pisałem musiał być natychmiastowo zrozumiały dla mnie i mojego szefa, który nie był starszym programistą Pythona i miał jeszcze mniej czasu niż ja. Chciał móc bardzo szybko usiąść do kodu i zmienić parę rzeczy od czasu do czasu, dlatego droga do rozszerzania kodu musiała być oczywista.

Kolejnym trudnym zadaniem było wybieranie co i jak testować. Testy miały oszczędzić nam czas przez wyeliminowanie potrzeby ręcznego sprawdzania czy wszystko działa po deploymencie nowego kodu. Czasem inwestowaliśmy sporo czasu w wypracowanie technik testowania w obszarach takich jak AWS Lambda i mikroserwisy, ponieważ nie było konkretnej literatury na ten temat. Ale nieraz po prostu nie było czasu na przetestowanie wszystkiego. Na szczęście, niektóre części systemu nie były „na widoku” użytkowników, a „zepsucie” ich na parę minut nie było wielkim problemem (ponieważ mieliśmy kolejki, backupy otrzymywanych wydarzeń, itd.). Tak więc jeśli cokolwiek poszło nie tak podczas wprowadzania zmian w tych częściach nasz system alarmów natychmiast nas o tym informował, a my od razu wprowadzaliśmy poprawki, przez co użytkownicy nie mogli zauważyć żadnych problemów.

2017.05 – 2017.10

Kontraktor Ansiblowy w ScyllaDB (https://www.scylladb.com/)
Zdalnie

Zostałem wynajęty aby stworzyć skrypty Ansiblowe do instalacji bazy danych ScyllaDB. Udało mi się stworzyć automatyzację instalacji pojedynczej instancji bazy. Później zacząłem pracę nad skryptami ustawiającymi cały klaster bazodanowy, ale ich nie dokończyłem. Byłem zbyt przepracowany posiadając ten kontrakt i etat w Iterio, więc zrezygnowałem z tego pierwszego.

Głównym wyzwaniem tutaj było wymaganie bazowania na istniejących, złożonych skryptach shellowych. Robiły takie rzeczy jak modyfikacja ustawień kernela dla maksymalizacji wydajności bazy. Problem polegał na tym, że zazwyczaj nie były idempotentne i były skore do modyfikowania stanu maszyny. Więc musiałem proponować zmiany w nich, a także przekomponować ich części w moje skrypty Ansiblowe.

2017.01 – 2017.06

Kontraktor od oprogramowania w Techno Service SA (https://tspcb.pl/) / Siled (http://siled.pl/)
Zdalnie, z paroma wizytami w fabryce w Gdańsku

Firma miała wielostopniowy, ręczny proces do tworzenia raportów z jakości lutów na produkowanych płytkach PCB. Początkowe analizy płytek były wykonywane przy pomocy wyspecjalizowanych maszyn przemysłowych. Zostałem zatrudniony do zautomatyzowania tego procesu.

Stary proces wyglądał tak:

- człowiek idzie do maszyny z pendrivem
- uruchamia na maszynie oprogramowanie zgrywające zestaw wyników testów
- kopiuje te dane na swój komputer
- uruchamia skrypt transformujący dane na arkusz Excela
- wysyła ten arkusz do osoby przygotowującej ostateczny raport
- ta druga osoba ręcznie kopiuje pola z wysłanego arkusza do arkusza ostatecznego raportu

Stworzyłem aplikację pod Windowsa dzięki której można było wybrać transze płytek którą użytkownik był zainteresowany, wprowadzić parę informacji i natychmiast dostać arkusz końcowego raportu. Było to możliwe, ponieważ maszyna do testowania PCB była podłączona do sieci i chodził na niej MS SQL Server zawierający dane z wszystkich testów płytek.

Użyte technologie:

- Python 3.6
- Tkinter, wybrany, ponieważ GUI było minimalne, a cała aplikacja musiała być bardzo łatwa do zainstalowania na Windowsie
- openpyxl do czytania i pisania arkuszy Excela
- PyAutoGUI do testowania GUI
- VirtualBox na którym chodził Windows XP, na którym chodził Microsoft SQL Server 2005 – wszystko po to, aby realistycznie testować testować wyciąganie danych z maszyny testującej PCB

2014.03 – 2016.04

Software Applications Developer w Intel Technology Poland
Gdańsk

Przykłady zadań w tej pracy, od ostatniego:

Prowadzenie zespołu programistów

Byłem szefem jednego z zespołów które pracowały nad projektem Trusted Analytics Platform (TAP). TAP był platformą PaaS przeznaczoną do tworzenia aplikacji analizujących dane (data analytics). Mój zespół głównie zajmował się integracją aplikacji tworzonych przez pozostałe zespoły, nadzorowaliśmy „buildy” i „deploymenty”, wypuszczanie nowych wersji, tworzyliśmy wewnętrzne narzędzia, oraz utrzymywaliśmy kilka mikroserwisów back-endowych.

Byłem odpowiedzialny za planowanie, estymację i koordynację pracy w zespole, a także za doszkalanie pozostałych członków zespołu. Co więcej:

- pełniłem funkcję pythonowego eksperta dla wszystkich zespołów projektu
- stworzyłem *mountepy* – bibliotekę Pythona pomagającą w izolacji mikroserwisów pod testy, oraz mającą za zadanie zwiększyć jakość naszego kodu
- wpływałem na metodyki wytwarzania we wszystkich zespołach
- egzekwowałem wymaganie licencyjne oprogramowania i komunikowałem się z prawnikami

Tworzenie mikroserwisów back-endowych dla TAP

Rozwijałem kilka mikroserwisów napisanych przy użyciu frameworka Spring Boot (Java), a także stworzyłem jeden w Pythonie (Flask). Pythonowy serwis zajmował się indeksowaniem i wyszukiwaniem danych, oparty był o ElasticSearch. Używaliśmy PaaSu CloudFoundry jako platformy dla naszych aplikacji. Czasem musieliśmy pogmerać w maszynach wirtualnych na których chodził klaster Cloud Foundry.

Utrzymanie bezpiecznego back-endu do zdalnych aktualizacji firmware'u (Intel Upgrade Service)

Mieliśmy dojrzały system rozproszony zawierający usługi back-endowe napisane w C#, front-end w ASP.NET, serwer i klienta w C++. Podstawą działania systemu był schemat podpisów cyfrowych EPID. Musiałem dostarczać patche do niemal wszystkich komponentów w warstwie aplikacji, oraz dokonywać deploymentów na dużej, geograficznie rozproszonej infrastrukturze opartej o Windows.

Musiałem także wykonywać testy, które dotykały wszystkiego od firmware'u na maszynie klienckiej, przez sterowniki i oprogramowanie warstwy aplikacji, a kończąc na wielowarstwowych usługach back-endowych z HSMami (Hardware Security Modules).

Implementacja bezpiecznych tuneli do wymiany certyfikatów przy użyciu sprzętowej kryptografii

Musiałem stworzyć aplikację w C++ 11 na osadzone platformy Linux, która mogła tworzyć, zarządzać i przesyłać certyfikaty X.509. Prywatne klucze certyfikatów oraz klucze używane do zabezpieczania połączeń były tworzone i przechowywane w sprzętowych modułach kryptograficznych (TPM). OpenSSL i TrouSerS były wykorzystane do tworzenia połączeń MTLS (TLS z uwierzytelnieniem obu stron).

Badanie wykorzystania kart SIM jako „elementów bezpiecznych" (secure element / SE)

Robiłem to sam jako moje pierwsze zadanie w zespole. Musiałem zaznajomić się z Java Micro Edition oraz sprzętem typu „smart card".

Okazuje się że niektóre karty SIM (i smart cards o formacie kart SIM) są w stanie wykonywać kryptografię asymetryczną, ale nie mogliśmy ich łatwo wykorzystać. Musielibyśmy uzyskać prawa operatora SIM, aby móc wysyłać nasz kod na karty przy

pomocy OTA, lub musielibyśmy zamówić transzę kart od producenta. Obie opcje nie były dla nas odpowiednie.

2011.05 – 2014.02

Test Engineer Intern at Intel Technology Poland
Gdańsk

Utrzymywałem i rozwijałem heterogeniczny (Windows, Linux, Android) rozproszony framework do zautomatyzowanych testów oprogramowania, firmware'u oraz sprzętu. Pisałem kod głównie w C# i Javie (zarówno wersja SE, jak i na Androida), ale pomagałem sobie pisząc skrypty w Pythonie.

Głównie skupiałem się na projektowaniu i implementacji wszechstronnej komunikacji w stylu RPC między aplikacjami na różnych platformach (wykorzystano WCF na Windowsie, JAX-WS na Linuxie, oraz mój własny protokół dla Androida). Współpracowaliśmy ściśle z naszym klientami – zespołami walidacyjnymi.

Rozwiązywanie niektórych problemów wymagało wiedzy na temat implementacji Javy i .NETa (np. niezgodności w implementacji socketów TCP na obu platformach).

Najważniejsze wystąpienia konferencyjne

TDD of Python microservices

EuroPython 2016. https://youtu.be/d-ka10jngQQ

Prezentacja narzędzi (niektórych stworzonych przeze mnie) i rozwiązań umożliwiających „Test-Driven Development" dla mikroserwisów napisanych w Pythonie. Przedstawiona z perspektywy osoby utrzymującej pojedynczy serwis.

Python microservices on PaaS done right

EuroPython 2015. https://youtu.be/WYXkpiaGBms

Zbiór wskazówek i praktyk pozwalających na stworzenie projektu opartego o mikroserwisy. Dotyczy wytwarzania, testowania oraz organizacji pracy.

Edukacja

2016.03 – 2016.05

Kurs „Cryptography I"
Stanford (przez platformę Coursera)

2012 – 2015

Magister inżynier informatyk
Politechnika Gdańska

Ogólny wynik – dobry plus
Specjalizacja w Inteligentnych Systemach Interaktywnych (AI, widzenie komputerowe, grafika)

Praca magisterska
Zdalne wywoływanie metod języka Java w systemie Android z platformy .NET
Opiekun pracy: dr inż. Jacek Lebiedź

Celem pracy było stworzenie zestawu bibliotek które umożliwiłyby wygodne zdalne wykonywanie kodu na Androidzie z platformy .NET. Cel został spełniony. Efektem ubocznym był patch do biblioteki *jsonrpc4j*, umożliwiający jej działanie na Androidzie.

2008 – 2012

Inżynier informatyk
Politechnika Gdańska

Ogólny wynika – dobry plus

Projekt końcowy
Aplikacja w C++ używająca OpenCV, która była w stanie uczyć się i rozpoznawać twarze przy pomocy metody wektorów własnych (eigenvector).

Języki

- Polski, ojczysty
- Angielski, płynny

Zainteresowania

- technologia i inżynieria w ogóle
- psychologia ewolucyjna
- mitologia (głównie słowiańska, nordycka i grecka)
- historia
- sztuki walki