1. Wstęp:

Stały rozwój metod programowania to nieodłączny element współczesnej branży informatycznej. Jeszcze kilka lat temu umiejętności programistyczne miały dużo większe znaczenie niż obecnie. Programista musiał skonfigurować swoje środowisko pracy oraz własnoręcznie kompilować i testować tworzoną przez siebie aplikację. Jednak nie były to jedyne problemy programistów z przeszłości. Niejednokrotnie można było usłyszeć historie w których duże firmy wykonując daną aplikacje napotykały etap, który wymagał weryfikacji stworzonego kodu. Moment ten był bardzo bolesnym krokiem podczas produkcji oprogramowania, ponieważ właśnie wtedy wychodziły na jaw wszystkie błędy w stworzonym kodzie. Proces integracji potrafił trwać miesiacami, a niekiedy nawet dłuzej niż produkcja samego kodu. Z biegiem czasu pojawiły się nowe metody rozwoju oprogramowania. Jedną z tych metod było programowanie zwinne. Wraz z narodzeniem się wspomnianej metody organizacji pracy wielu programistów powstało wiele sposobów na ułatwienie życia osób tworzących aplikacje. Ułatwieniem tym był rozwój takiej branży jak ciągła integracja. Ciągła integracja to praktyka tworzenia oprogramowania, w której członkowie zespołu często sprawdzają swoją pracę. Prowadzi to do co najmniej kilku weryfikacji oprogramowania dziennie. Każda zmiana w kodzie jest weryfikowana przez zautomatyzowaną kompilację oraz testy, aby jak najszybciej wykryć błędy w kodzie. Wiele zespołów odkrywa, że takie podejście prowadzi do zmniejszenia problemów z integracją i pozwala zespołowi szybciej opracowywać spójne oprogramowanie. Dzięki tej elastycznej metody prowadzenia projektu ryzyko niedotrzymania terminu wykonania aplikacji jest niwelowane do minimum. O ile praktyka ta nie wymaga do wdrożenia żadnego narzędzia to dobrym pomysłem jest użycie tak zwanego serwera ciągłej integracji. Obecny rynek jest przepełniony takowymi narzędziami zarówno w formie komercyjnej jak i open source. Najczęściej spotyka się jednak w użyciu takie serwery jak Jenkins, Gitlab Ci, CircleCi, TeamCity, Travis CI. Oprócz wymienionych serwerów istnieje jednak wiele innych alternatyw. Stworzenie skutecznej oraz odpowiednio skonfigurowanej infrastruktury ciągłej integracji nie należy jednak do prostych zadań i wymaga odpowiedniej wiedzy z wielu zagadanień współczesnej informatyki. W skład owych zagadnień wchodzą serwery ciągłej integracji, systemy kontroli wersji, metody monitorowania infrastruktury, rozwiązania chmurowe oraz sieciowe. Ponadto stworzoną infrastrukturę należy odpowiednio dopasować do realizowanego w danym momencie zadania. W niniejszej pracy postaram się opisać odpowiednie kryteria towarzyszące doborowi odpowiednich technologii do określonego projektu oraz zaimplementować i porównać prezentowane rozwiązania.

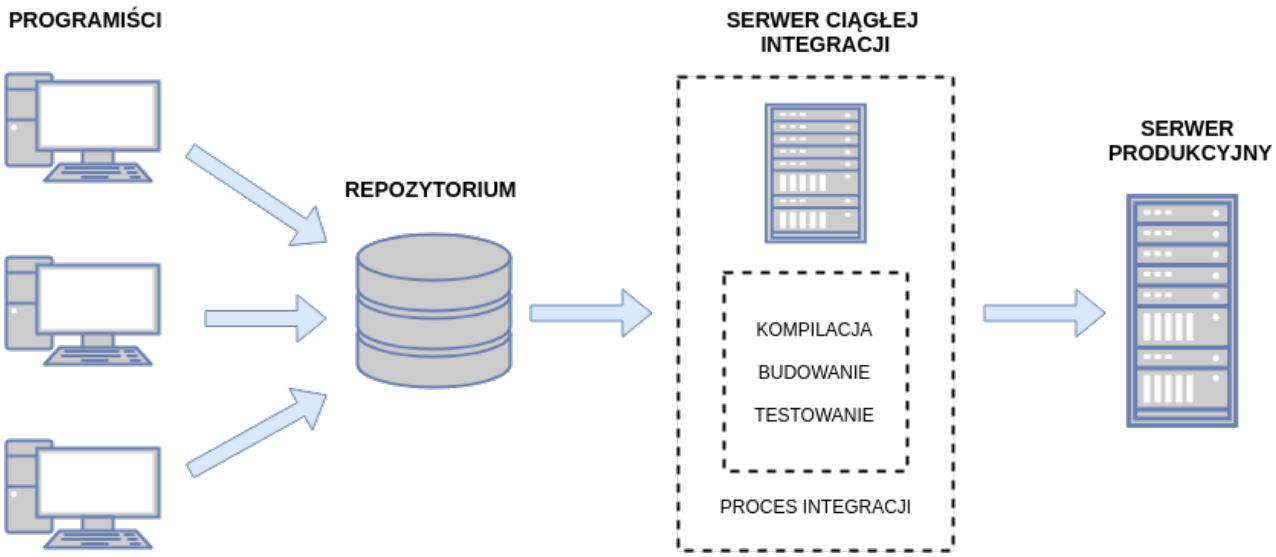
2. Cel i zakres

Zespoły odpowiedzialne za stworzenie i administrowanie infrastruktury ciągłej integracji często stoją przed wieloma wyzwaniami, takimi jak wybór odpowiednich technologii, dobór odpowiednich zasobów, automatyzacja wykonywanych czynności, monitorowanie infrastruktury oraz optymalizacja procesu wytwarzania oproramowania. Praca takich zespołów to seria niekończących się wyzwań oraz problemów którym trzeba sprostać. Celem tej pracy jest przegląd technologii jakie można wykorzystać podczas implementacji środowiska ciągłej integracji, wyznaczenie kryteriów pozwalających na optymalny dobór dostępnych zasobów obliczeniowych w zależności od wielkości projektu programistycznego oraz porównanie zaproponowanych rozwiązań. Porównaniu będzie podlegać infrastruktura, w której serwer ciagłej integracji oraz system kontroli wersji działają jako osobne serwisy oraz infrastruktura zbudowana w oparciu o narzędzie zawierające te elementy w jednym serwisie. Do realizacji zadania wymagającego wykorzystania kilka serwisów zostanie wykorzystany Jenkins jako serwer ciagłej integracji, Gerrit jako system kontroli wersji oraz narzędzie odpowiedzialne za dynamiczne zarządzanie wykonywanymi testami jakim jest Zuul. Druga infrastruktura będzie zbudowana w oparciu o bardzo obszerne narzędzie jakim jest Gitlab. Rozproszenie budowania i testowania oprogramowania zostanie wykonane przy pomocy OpenStack’a, czyli oprogramowania z dziedziny chmur obliczeniowych. Ważnym elementem pracy jest wskazanie dobrych praktyk podczas implementacji takich rozwiązań. Przykładem dobrych praktyk jest automatyzacja czynności wykonywanych podczas zarządzania przygotowaną infrastrukturą, utrzymywanie konfiguracji serwerów ciągłej integracji w postaci kodu oraz dokładne monitorowanie stworzonego środowiska. Do automatyzacji rutynowych czynności takich jak tworzenie nowych instancji w chmurze lub instalowanie pakietów potrzebnych do wykonania testów, wykorzystane zostanie bardzo rozbudowane narzędzie jakim jest Ansible. Nieodłącznym elementem każdego środowiska ciągłej integracji jest dokładne monitorowanie wykorzystanych zasobów oraz serwisów. Realizacja takiego monitoringu zostanie wykonana przy pomocy dwóch narzędzi. Pierwszym z nich jest Prometheus, którego zadanie polega na zbieraniu informacji na temat każdej maszyny wirtualnej wykorzystywanej w stworzonej infrastrukturze oraz przekazywanie zebranej wiedzy do serwisu odpowiedzialnego za wyświetlenie zebranych danych. Serwisem tym jest Grafana. Podczas realizacji pracy zostaną wykonane badania mające na celu sprawdzenie jak wielkość projektu wpływa na zużycie dostepnych zasobów obliczeniowych, zasobów dyskowych oraz ruchu sieciowego. Przeprowadzone badania mają na celu wyznaczenie pewnych standardów pomocnych przy doborze zasobów i narzędzi w zależności od wykonywanego projektu.

3. Aspekt inżynierski

3.1 Serwery ciągłej integracji

Aby zrozumieć czym właściwie jest ciągła integracja, należy przedstawić problem, który można rozwiązać właśnie przy jej pomocy. Przykładem takiego problemu może być sytuacja, w której dwóch programistów pracuje równocześnie nad wdrożeniem dwóch różnych funkjonalności. Zrealizowanie tych zadań wymaga od nich modyfikacji wielu plików wymaganych do poprawnego działania zarówno jednej funkjonalności jak i drugiej. Po kilku tygodniach pracy może się jednak okazać, że o ile osobno dane funkcjonalności działają poprawnie to przy próbie użycia obu naraz pojawia się problem. Problem ten wynika z tego, że zmodyfikowano wiele zależnych od siebie plików i obie funkcjonalności wymagają wzajemnej integracji. Proces takiej integracji trwa zazwyczaj bardzo długo, a momentami wydaje się niemożliwy do wykonania. Czasem też lepszym rozwiązaniem jest napisanie obu funkcjonalności od nowa z uwzględnieniem zauważonych wcześniej błędów. Właśnie w tym celu wymyślono takie narzędzia jak serwery ciągłej integracji. Zadaniem wspomnianych serwerów jest kontrolowanie, budowanie oraz testowanie wytwarzanego oprogramowania w sposób ciągły, tak aby wykrywać ewentualne błędy w trakcie wytwarzania oprogramowania.



Rys. 1 Przykład podstawowej architektury infrastruktury realizującej ciągłą integracje

Standardowa infrsatruktura ciągłej integracji składa się z takich elementów jak system kontroli wersji z repozytorium do którego mają dostęp programiści realizujący dany projekt, serwera ciągłej integracji oraz serwera produkcyjnego. Przykład takiej infrastruktury został przedstawiony na rys. 1. Takie rozwiązanie doskonale spełnia swoją funkcje przy małych aplikacjach, w które zaangażowana jest niewielka grupa programistów. Dla większych projektów dodatkowym wymaganiem jest jednak bardziej rozbudowane środowisko umożliwiające integracje na szerszą skalę. Przykładowo w przypadku dużej ilości zmian wprowadzonych do repozytorium pojedyńcza maszyna nie byłaby w stanie przeprowadzić testów wystarczająco szybko. Serwery ciągłej integracji pozwalają jednak na użycie większej ilości maszyn fizycznych bądź wirtualnych w celu zbudowania i przetestowania oprogramowania dla większej ilości wprowadzonych zmian. Do tego celu bardzo często wykorzystuje się rozwiązania chmurowe, które cieszą się coraz większą popularnością. Dzięki rozszerzeniu infrastruktury o ten element można przeprowadzać równoległe kompilacje na wielu niezależnych od siebie maszynach.

3.2 Rozwiązania chmurowe

Rozwiązania chmurowe stają sie coraz bardziej popularnym modelem świadczenia usług IT. Samo pojęcie chmury obliczeniowej jest ściśle związane z wirtualizacją. Dzięki korzystaniu z usług firmy umożliwiających dostęp do maszyn wirtualnych nie ma konieczności instalowania oraz administrowania własnej infrastruktury. Ponadto wykorzystanie w swoich projektach chmury zwiększa bezpieczeństwo. Jest to spowodowane tym, że dane są szyfrowane oraz cały ciężar administracji maszynami serwerowymi zostaje przeniesiony na zewnętrzną usługę IT. Dostęp do maszyn produkcyjnych mają jedynie komputery klienckie np. programiści biorący udział w realizacji danego projektu bądź administratorzy infrastruktury ciągłej integracji. Dobrymi przykładami chmur obecnie wystepujących na rynku są Amazon Web Services, Microsoft Azure, Google Cloud oraz OpenStack.

Chmury obliczeniowe pełnią szczególnie ważną rolę podczas tworzenia infrastruktury ciągłej integracji dla większych projektów programistycznych. Projekty te wymagają wykorzystania większych zasobów, ponieważ zmiany w wytwarzanym oprogramowaniu są znacznie częstrze oraz testom podlega wieksza ilość kodu. Szybki rozwój technologii spowodował, że komputery przy których obecnie pracują programiści mają dużą moc obliczeniową i dobrze spełniają swoje zadanie podczas programowania. Nie zmienia to jednak faktu, że komputery na których programiści tworzą oprogramowanie oprócz kompilacji wykonują sporo innych czynności. Zasoby dostarczane przez usługi chmurowe, mają tylko jeden cel, którym jest kompilacja oraz przetestowanie stworzonej części kodu. Wirutalne maszyny pracujące w infrastrukturze integracji kodu bardzo często mają dużo większe zasoby oraz moc obliczeniową niż komputery na których pracują programiści. Dzięki temu kompilacja danego programu na lokalnym komputerze może być dużo wolniejsza niż kompilacja tego samego programu w chmurze.

3.3 Systemy kontorli wersji

==========TO DO==========

3.4 Monitorowanie infrastruktury

==========TO DO==========

3.5 Infrastruktura ciągłej integracji z wykorzystaniem Jenkinsa

==========TO DO==========

3.5.1 Jenkins

==========TO DO==========

3.5.2 Zuul

==========TO DO==========

3.5.3 Gerrit

==========TO DO==========

3.5.4 Opis infrastruktury

==========TO DO==========

3.6 Infrastruktura ciągłej integracji z wykorzystaniem Gitlaba

==========TO DO==========

3.6.1 Gitlab

==========TO DO==========

3.6.2 Opis infrastruktury

==========TO DO==========

3.7 Monitorowanie infrastruktury z wykorzystaniem Grafany

==========TO DO==========

3.8 Metody zarządzania infrastrukturą

==========TO DO==========

4. Aspekt badawczy

5. Wnioski

6. Literatura

[1] Shahin, Mojtaba & Zahedi, Mansooreh & Ali Babar, Muhammad & Zhu, Liming. (2018). An empirical study of architecting for continuous delivery and deployment. Empirical Software Engineering. 10.1007/s10664-018-9651-4

[2] Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, and Bogdan Vasilescu. 2017. The impact of continuous integration on other software development practices: a large-scale empirical study. In Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017). IEEE Press, Piscataway, NJ, USA, 60-71.

[3] Ståhl, Daniel & Bosch, Jan. (2014). Modeling continuous integration practice differences in industry software development. Journal of Systems and Software. 87. 48–59. 10.1016/j.jss.2013.08.032.

[4] Cloud Computing Bible, Barrie Sosinsky, 2011 Willey Publishing, Inc ISBN: 978-0470903568

[5] Continous Integration, Martin Fowler, 01 May 2006

[6] Ansible: Up and Running, Lorin Hochstein, Rene Moser, 20 Jul 2017 O'Reilly Media, Inc, ISBN: 978-1491979754

[7] The Linux Command Line, William E. Shotts Jr., 2015 Helion, ISBN: 978-1593273897

[8] https://jenkins.io/doc/

[9] https://gerrit-review.googlesource.com/Documentation/

[10] https://about.gitlab.com/install/?version=ce

[11] https://zuul-ci.org/docs/zuul/

[12] https://docs.ansible.com/

[13] https://docs.openstack.org/install-guide/overview.html

[14] http://docs.grafana.org/

[15] https://prometheus.io/docs/