Plan CI/CD

rev 2.0

Pipeline Continuous Integration | Continuous Deployment

Jakub Szumilas

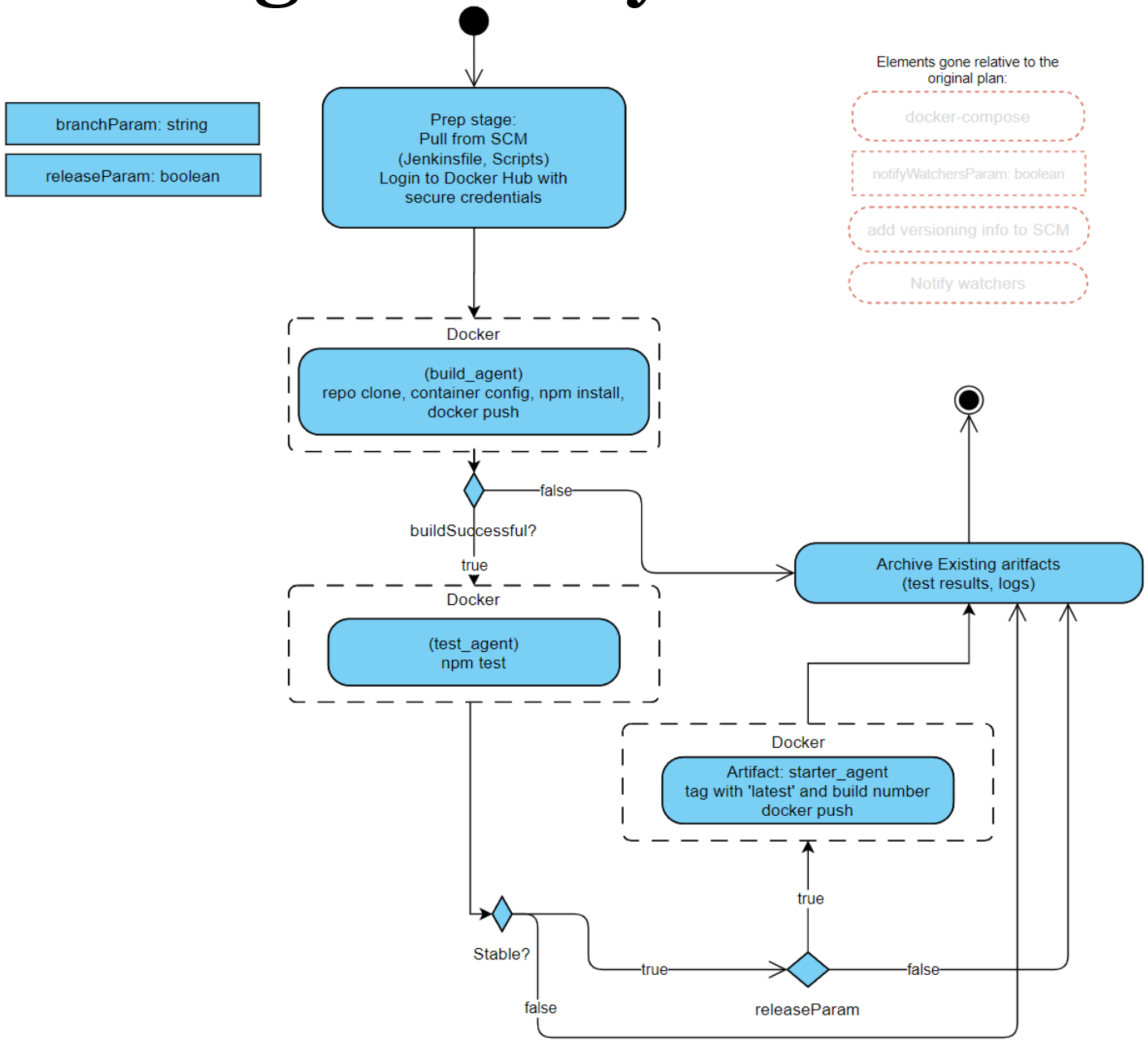
27.05.2021

Wykorzystane technologie

- Docker
  - DockerHub
  - ~~*Docker Compose*~~
- Jenkins
  - Groovy
  - Jenkinsfile
  - cURL, Docker Plugin, cron
- Git
  - GitHub
  - Git Hooks

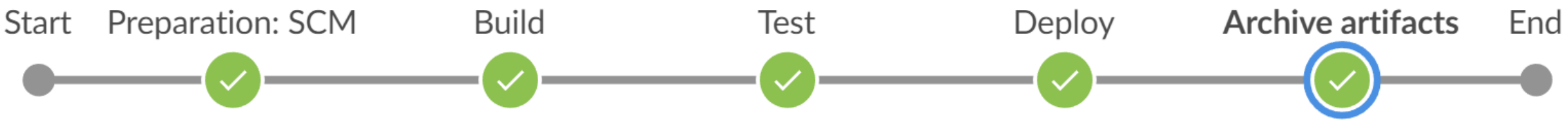
- Aplikacja: node-chat-app
  - JavaScript ES6
  - NodeJS
  - Express
  - Mocha, Expect
  - Socket.io
  - Mustache.js
  - Moment.js

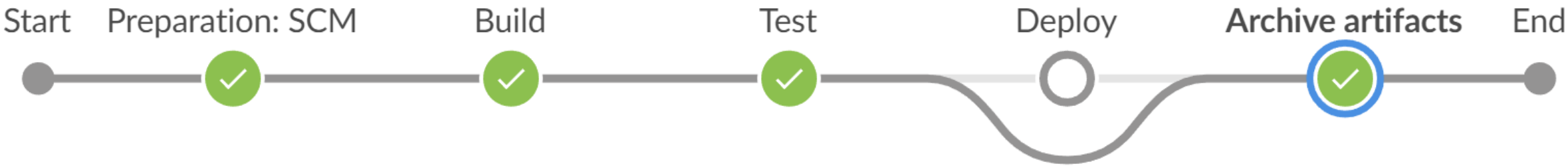
Diagram aktywności



- Za cały proces odpowiada agent Jenkins, który uruchomiony jest na kontenerze Docker.
  - Detale działania opisane w Jenkinsfile
- Build uruchamiany w ramach pojedynczego pipeline'a podzielonego na stage i stepy.
- Trigger pipeline'u: wykonanie commita przez developera, każdego dnia około godziny 3:40 lub manualnie, z podaniem konkretnej gałęzi lub wybraniem opcji release'u.
- Jenkins pobiera potrzebne komponenty i nimi operuje.
- Procesy buildu i testów wykorzystuje skrypty konfiguracyjne *npm* i odbywają się w osobnych kontenerach Docker.
- W zależności od wyniku buildu i testów, przejście do releasu:
  - Release – upublicznienie obrazu aplikacji czatu w Docker Hub
  - Odpowiednie otagowanie upublicznionego obrazu numerem builda.
- Zachowanie artefaktów: wyniku testów i logów z pipeline'a/buildu
- Osoby obserwujące mogą uruchomić powiadomienia Jenkinsa w przeglądarce.
- Aby jeszcze bardziej zautomatyzować proces deploymentu, wystarczy ustawić domyślny parametr release'u na *true* (należy wtedy utrzymywać rygorystyczne testy) oraz filtrowanie po gałęzi (tylko dla gałęzi releasowej, np. *master*)

Build stages:

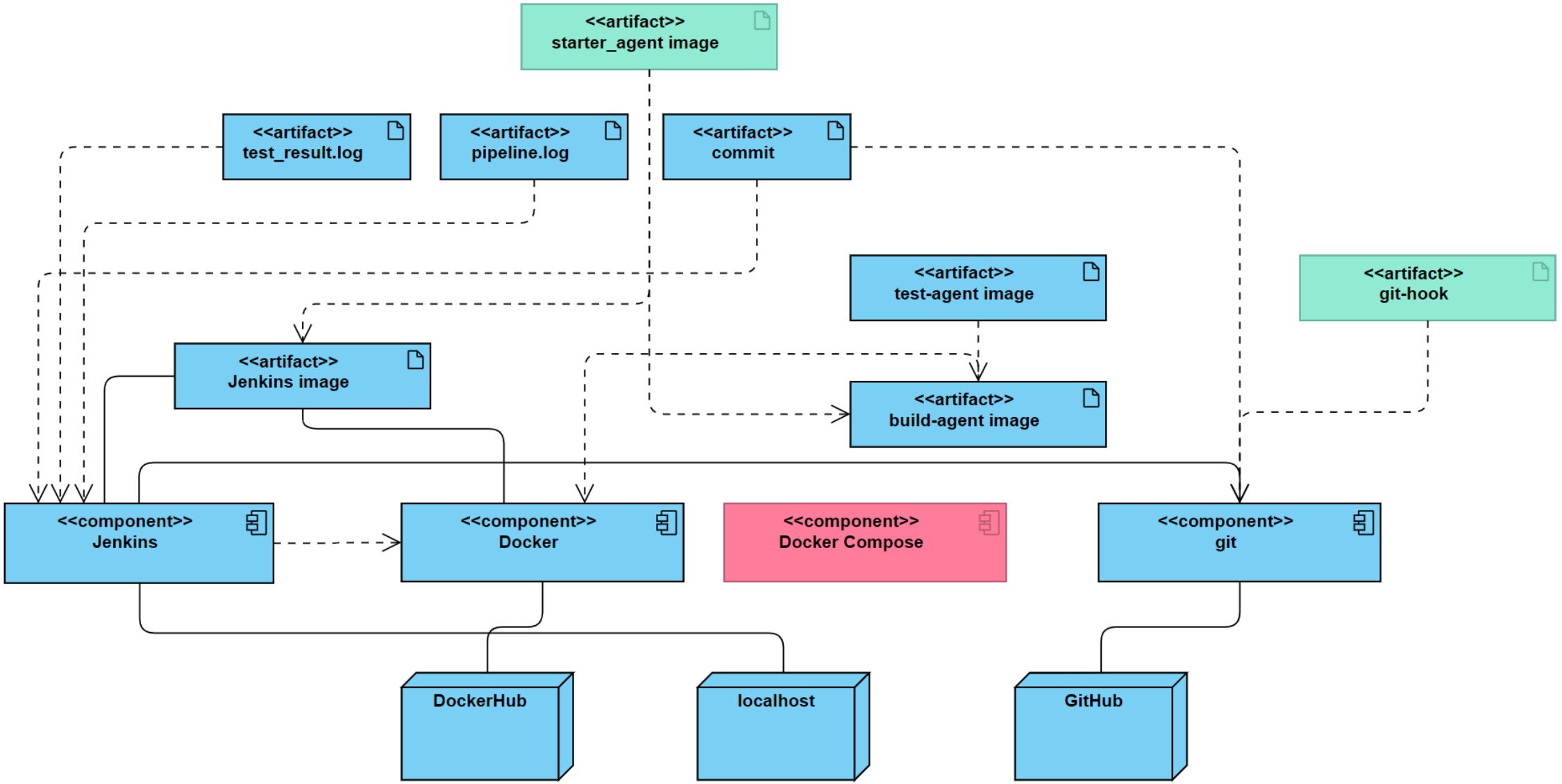




Tabelaryczne przedstawienie

Krok	Technologia	Link do pliku konfiguracyjnego	Nr linii	Komentarz
Preparation	GitHub, Jenkins Credentials	Jenkinsfile	23	Pobieranie Jenkinsfile z SCM, logowanie do Docker Huba.
Build	Docker, NPM, Docker Hub	Jenkinsfile	34	Biorąc pod uwagę dynamiczną naturę JavaScript, nie jest to typowy „build” (proces kompilacji), raczej zarządzanie zależnościami i doinstalowanie ich braków, oraz co najważniejsze, ten agent ma na celu maksymalne uproszczenie postawienia środowiska uruchomieniowego. Agent buildujący jest pushowany do Docker Huba. W początkowej wersji planowane było wykorzystanie Docker-compose, ale zrezygnowano z tego dla elastyczności w operowaniu działaniem z poziomu Jenkinsfile’a.
		Dockerfile	1	
Test	NPM, Mocha, Docker	Jenkinsfile	49	Testy wg. planu, tylko poza docker-compose.
		Dockerifle	1	
Deploy	Docker, Docker Hub	Jenkinsfile	62	Upublicznienie obrazu startującego na Docker-hub z odpowiednim tagiem.
		Dockerfile	1	
Archive Artifacts	NPM, Shell, Docker	Jenkinsfile	78	Zebranie plików ze środowiska, które zostają zarchiwizowane dla danego builda.

Diagram wdrożeniowy



Tabelaryczne przedstawienie

Artefakt	Technologia	Link do pliku	Komentarz
Jenkinsfile	Groovy, JVM, Jenkins, GitHub	Jenkinsfile	Według planu. Główna część konfiguracji procesu.
test_results.log	(text file) mocha, npm	*(localhost)*	Rezultaty testu
pipeline.log	(text file) Jenkins	*(localhost)*	Jak wyżej
szumied/build_agent	Docker Hub	Docker Hub	Obraz ułatwiający konfigurację środowiska potrzebną do uruchomienia programu. Ogranicza potrzebę instalowania dodatkowego software'u.
szumied/node_chat_starter	Docker Hub	Docker Hub	Końcowy artefakt, który ma być kierowany do użytkowników. Działa. ☺
Dockerfile: build_agent	Dockerfile	Dockerfile	Dockerfile odpowiadający za konfigurację jednego z powyższych agentów. Według planu.
Dockerfile: test_agent	Dockerfile	Dockerifle	Dockerfile odpowiadający za konfiguracje agenta testowego. Wg. Założeń.
Dockerfile: starter_agent	Dockerfile	Dockerfile	Dockerfile odpowiadający za konfigurację obrazu startowego. W początkowym planie zakładano możliwość różnego rodzaju releasu (np. opublikowanie obrazu lub dostarczenie go na serwer gdzieś i uruchomienie)
Kod źródłowy: node-chat-app	Node.JS, Express, GitHub	binhxn/node-chat-app	Bez zmian.