Soluzione progetto

# Fase 1

* docker pull jenkins/jenkins

Immagine che contiene testo

Descrizione generata automaticamente

* docker run -d --name jenkins -p 8080:8080 -v %PROJECT\_HOME%\progetto\jenkins\_home:/var/jenkins\_home jenkins/Jenkins
* Inserire l’admin password: Immagine che contiene testo, elettronico, screenshot, schermo

  Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene tavolo

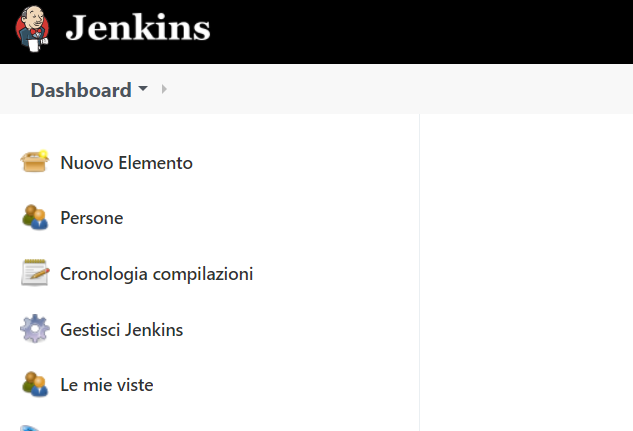
Descrizione generata automaticamente

Immagine che contiene tavolo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

* Per compilare applicazioni angular mi serve NodeJS

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

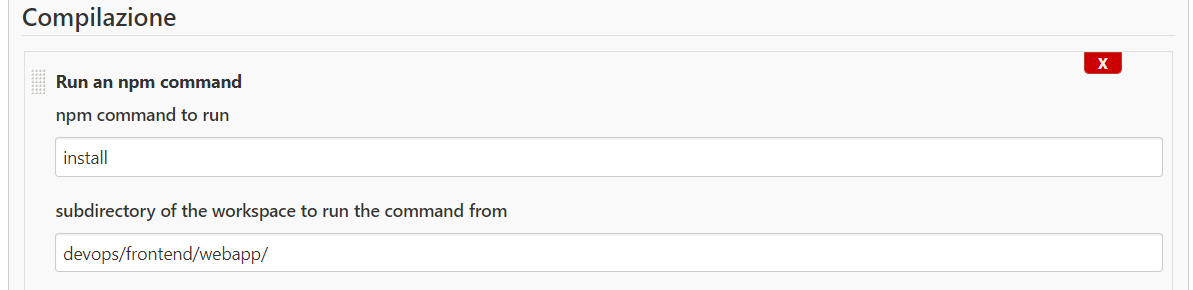
Immagine che contiene testo

Descrizione generata automaticamente

* Ora posso usare NPM; faccio prima l’install dei node\_modules

Immagine che contiene testo, screenshot, portatile

Descrizione generata automaticamente



* Quindi con lo stesso tipo di step eseguo la build dell’app angular

Immagine che contiene testo

Descrizione generata automaticamente

* Per eseguire la compilazione dell’applicazione Spring Boot, sfrutto la distribuzione di maven già presente nelle applicazioni Spring Boot (maven wrapper):

Immagine che contiene tavolo

Descrizione generata automaticamente

* La compilazione fallisce!!
* Dato che il progetto Boot è in Java 11 e Jenkins funziona con una Java 8, devo installare sul build serve la JDK adatta alla compilazione; potrei installare nel container la JDK 1, ma mi conviene definire una nuova immagine di build server con installato ciò che mi serve; quindi definisco un Dockerfile con le info necessarie (fino a qui) e faccio la build della nuova immagine:

FROM jenkins/jenkins

USER root

RUN apt update

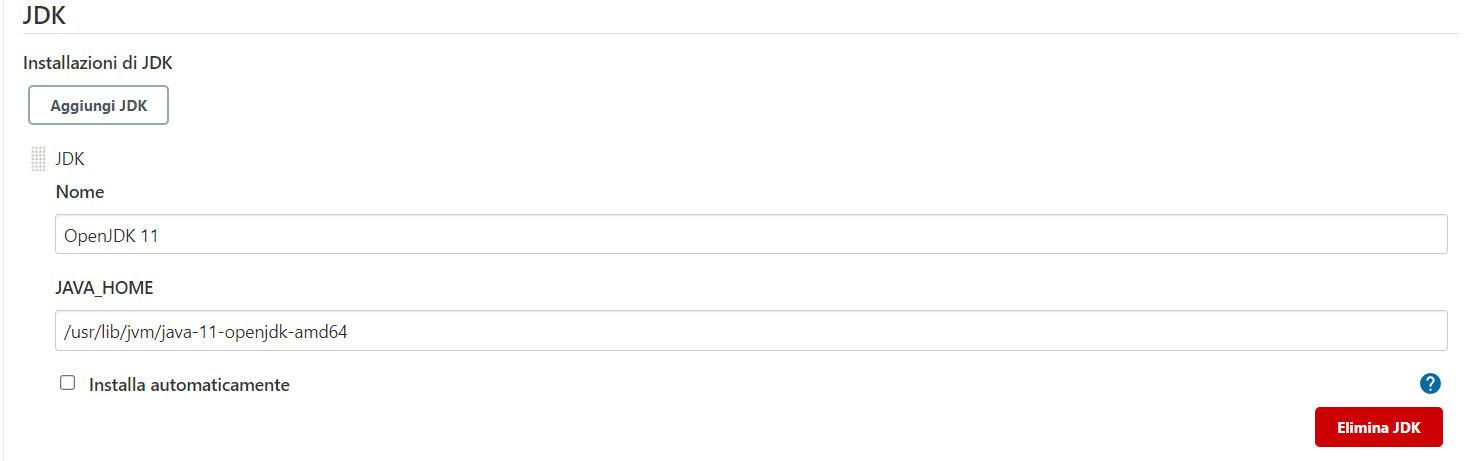
RUN apt -y install openjdk-11-jdk

USER jenkins

* Eseguo il comando di build, creando anche un tag della nuova immagine: docker build -t ipzs\_jenkins .
* Ora posso eseguire un nuovo container con la nuova immagine

docker run -d --name jenkins -p 8080:8080 -v %PROJECT\_HOME%\progetto\jenkins\_home:/var/jenkins\_home ipzs\_jenkins

* Ora devo istruire Jenkins in modo che trova il nuovo JDK 11 nella “Configurazione Global Strumenti”:



* Modifico lo step di compilazione per compilare con JDK 11

Immagine che contiene testo

Descrizione generata automaticamente

* Aggiungo la pubblicazione dei report JUnit nelle azioni di post compilazione

Immagine che contiene testo

Descrizione generata automaticamente

* Devo zippare gli artifacts e poi archiviarli



Immagine che contiene testo

Descrizione generata automaticamente

* Per semplicità, devo dare accesso anonimo agli artifacts archiviati

Immagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamente

* Possiamo ora definire i Dockerfile dei vari servizi componenti
  + Frontend: devo prendere dist.zip da Jenkins, scompattarlo e spostarlo nella directory di default di NGINX

FROM nginx

RUN apt update

RUN apt install -y wget unzip

RUN wget http://192.168.1.51:8080/job/progetto-sessione-1/lastSuccessfulBuild/artifact/devops/frontend/webapp/dist/dist.zip && unzip dist.zip

RUN rm dist.zip

RUN cp /dist/webapp/\* /usr/share/nginx/html

RUN rm -r /dist

* + Per buildare e taggare l’immagine: ***docker build -t ipzs\_frontend .***
  + Backend: devo prelevare il JAR da Jenkins, copiarlo localmente e definire un entry ponit per lanciarlo

FROM openjdk:11.0.10

RUN wget http://192.168.1.51:8080/job/progetto-sessione-1/lastSuccessfulBuild/artifact/devops/eclipse/progetto-finale/target/progetto-finale-0.0.1-SNAPSHOT.jar && cp progetto-finale-0.0.1-SNAPSHOT.jar app.jar

RUN rm progetto-finale-0.0.1-SNAPSHOT.jar

ENTRYPOINT ["java","-jar","/app.jar"]

* + Per buildare e taggare l’immagine: ***docker build -t ipzs\_backend .***

NB: se cambiano o dist.zip o il .jar, devo rifare la build senza usare la cache (opzione  ***--no-cache***)

Una volta definite le due immagini (back e front) posso utilizzare un docker compose per realizzare l’intera infrastruttura; nell’infrastruttura includo anche un db mysql per le operazioni di persistenza:

version: "3.9"

services:

    frontend:

        build: ./container-frontend

        ports:

            - 8000:80

    backend:

        build: ./container-backend

        ports:

            - 8001:9090

        depends\_on:

            - db

    db:

        image: mysql

        environment:

            MYSQL\_ROOT\_PASSWORD: passw0rd

        ports:

            - 3306:3306

        volumes:

            - "[PROJECT\_ROOT]\\progetto\\dati:/var/lib/mysql"

NB: i dati del db mysql risiederanno in una directory locale sul docker host.

Quindi posso avviare l’intera infrastruttura con: ***docker-compose up -d***

NB: le stringhe di connessione su ambiente locale e nell’infrastruttura docker, potrebbero essere differenti; in questo caso ci può essere utile fare un replacement nei file di configurazione.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente