Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

Generazione automatica di file YAML per il progetto FREEDA

Relatore:
Prof. AMADINI ROBERTO

Presentata da:
LUCA ANGHINOLFI

Correlatore:
Dott.GAZZA SIMONE

Sessione Unica
Anno Accademico 2023/2024

Indice

Capitolo 1

Introduzione

Il progetto FREEDA (Failure-Resilient, Energy-Aware, and Explainable Deployment of Microservice-based Applications over Cloud-IoT infrastructures) è un progetto di ricerca innovativa e avanzata, parte del programma PRIN (Progetti di Ricerca di Rilevante Interesse Nazionale), in collaborazione con l'Università di Pisa e il Politecnico di Milano. FREEDA nasce per affrontare le sfide che provengono dall'evoluzione dei sistemi tecnologici distribuiti e dall'integrazione delle infrastrutture Cloud e dell'Internet of Things (IoT), con un sistema che collega il Cloud ai dispositivi di rete periferici, chiamati Edge.

Con l'aumento dei dispositivi connessi e della capacità di calcolo distribuita, FREEDA risponde al bisogno di strumenti per distribuire le applicazioni basate su microservizi (MSA) nelle infrastrutture Cloud-IoT in modo adattivo. I microservizi permettono di creare sistemi modulari e scalabili che si adattano in tempo reale alle esigenze degli utenti e alle condizioni della rete. FREEDA punta a sviluppare metodi di distribuzione che siano affidabili, efficienti dal punto di vista energetico e in grado di spiegare chiaramente le scelte fatte nella gestione delle applicazioni.

Il contesto di questa ricerca è dovuto alla crescente diffusione dei dispositivi IoT e al miglioramento continuo della capacità di calcolo dei dispositivi

connessi. Questi sviluppi richiedono che le infrastrutture Cloud evolvano verso ambienti distribuiti e pervasivi, dove il calcolo non avviene solo nei datacenter centrali, ma anche ai margini della rete, utilizzando i dispositivi Edge. Questo approccio offre un maggiore risparmio energetico e affronta importanti sfide per quanto riguarda la resilienza ai guasti e la gestione della complessità.

Un elemento centrale di FREEDA è l'uso di tecniche di continuous reasoning e constraint reasoning, che permettono al sistema di monitorare e migliorare la configurazione dei microservizi in tempo reale in caso di guasti. Grazie a queste tecniche, la distribuzione diventa dinamica e si adatta ai cambiamenti della rete, del carico di lavoro e della disponibilità delle risorse, garantendo una gestione ottimale delle applicazioni anche in situazioni complesse e dinamiche.

In conclusione, FREEDA si propone come una soluzione all'avanguardia per automatizzare e migliorare la gestione delle MSA (Microservice based Application) su infrastrutture Cloud-IoT.

1.1 Obiettivo della tesi

In questa tesi verrà presentata la creazione del pannello grafico per la generazione automatica del file YAML da sottoporre al modello FREEDA. In particolare, l'obiettivo è stato quello di sviluppare un'interfaccia utente basata su **JavaFX** [JavaFX] , in grado di raccogliere e organizzare i dati inseriti dall'utente in modo strutturato. I dati acquisiti tramite il pannello vengono elaborati e convertiti in un file YAML utilizzando la libreria **SnakeYAML** [SnakeYAML], che permette di serializzare oggetti Java in formato YAML in maniera efficiente.

Il processo di generazione del file YAML prevede diversi passaggi:

1. **Acquisizione dei dati** L'utente interagisce con il pannello JavaFX per definire i parametri relativi ai componenti, alle infrastrutture e alle risorse necessarie per il modello FREEDA.

2. **Elaborazione e strutturazione** I dati raccolti vengono organizzati in strutture dati adeguate all'interno dell'applicazione, garantendo la corretta associazione tra i diversi elementi.

3. **Serializzazione con SnakeYAML** Una volta strutturati, i dati vengono trasformati in un file YAML conforme ai requisiti del modello FREEDA, assicurando che la sintassi e la gerarchia delle informazioni siano corrette.

4. **Generazione dei file per componenti, infrastrutture e risorse** Per facilitare l'integrazione con FREEDA, i dati vengono suddivisi in più file YAML, ciascuno dedicato a una specifica categoria (componenti, infrastrutture, risorse). Questo approccio garantisce modularità e facilita eventuali modifiche o aggiornamenti futuri.

L'utilizzo di JavaFX [JavaFX] ha permesso di creare un'interfaccia intuitiva e dinamica, mentre l'integrazione con SnakeYAML ha reso possibile la generazione di file YAML in modo flessibile ed efficiente. Il risultato è un sistema che automatizza il processo di creazione della configurazione per il modello FREEDA, riducendo il rischio di errori manuali e migliorando la produttività. Inoltre, il sistema si estende al deployment: i file YAML generati vengono passati a un parser che li interpreta per avviare automaticamente il deploy dell'infrastruttura e delle applicazioni.

Capitolo 2

Background

2.1 Panoramica

2.1.1 Evoluzione delle infrastrutture Cloud-IoT

Negli ultimi anni, l'integrazione tra le tecnologie Cloud e l'Internet of Things (IoT) ha modificato il modo in cui i dati vengono raccolti, elaborati e distribuiti.

Internet of Things (IoT) è un paradigma tecnologico che consente la connessione e l'interazione di dispositivi fisici. Questi dispositivi, chiamati *smart objects*, includono sensori, attuatori, elettrodomestici intelligenti, macchinari industriali, veicoli connessi e molti altri sistemi in grado di generare e scambiare dati. L'IoT permette di monitorare e controllare processi fisici in tempo reale, abilitando nuove applicazioni in diversi settori, come la domotica, la sanità, l'industria e le smart cities.

Cloud Computing e Cloud-IoT è un modello di elaborazione che fornisce risorse computazionali (come server, storage e database) su richiesta, senza la necessità di gestione diretta da parte dell'utente. Questo paradigma

ha reso possibile la scalabilità e la flessibilità necessarie per gestire grandi quantità di dati generati dall'IoT.

L'integrazione tra Cloud e IoT ha dato origine al concetto di Cloud-IoT, ovvero un'architettura in cui i dispositivi IoT inviano i dati a piattaforme cloud per l'elaborazione, l'analisi e la conservazione. Questa integrazione presenta diversi vantaggi:

- **Scalabilità**: il cloud consente di gestire enormi quantità di dati in modo dinamico, adattandosi alle esigenze dell'infrastruttura IoT.

- **Efficienza computazionale**: le risorse cloud offrono capacità di calcolo avanzate, permettendo l'analisi di dati complessi e l'esecuzione di algoritmi computazionalmente costosi, ad esempio algoritmi di intelligenza artificiale.

- **Accessibilità**: l'uso del cloud consente di accedere ai dati e ai servizi da qualsiasi luogo, facilitando l'integrazione tra diversi dispositivi e piattaforme IoT.

L'evoluzione delle architetture informatiche ha portato all'evoluzione del modello cloud centralizzato a favore di soluzioni ibride, che includono Edge Computing e Fog Computing. Queste tecnologie permettono l'elaborazione dei dati direttamente nei nodi più vicini alla loro origine, riducendo la latenza e ottimizzando il traffico di rete. L'utilizzo di queste architetture ha migliorato le prestazioni dei sistemi IoT, consentendo un'elaborazione più efficiente e reattiva.

2.1.2 Paradigmi e Metodi Innovativi

Il progetto FREEDA si basa su un insieme di tecnologie avanzate per affrontare le sfide legate alla distribuzione di applicazioni basate su microservizi

(MSA) su infrastrutture Cloud-IoT. L'obiettivo è garantire affidabilità, efficienza energetica e adattabilità dinamica delle applicazioni distribuite. Tra le principali metodologie adottate, spiccano il *Continuous Reasoning* e il *Constraint Reasoning*, che permettono una gestione intelligente delle risorse e un'ottimizzazione continua delle configurazioni di sistema. Inoltre, FREEDA sfrutta il paradigma delle *Microservice-based Applications* per migliorare la modularità e la scalabilità dell'infrastruttura.

Continuous Reasoning e Constraint Reasoning è una metodologia che consente di effettuare valutazioni e aggiornamenti continui sulle configurazioni del sistema, adattandosi dinamicamente ai cambiamenti dell'ambiente e ai nuovi requisiti. Questo approccio è essenziale per garantire che le decisioni di allocazione e distribuzione delle risorse rimangano ottimali anche in presenza di variazioni nel carico di lavoro o nella disponibilità delle risorse.

Il Constraint Reasoning, invece, è un paradigma basato sulla definizione e risoluzione di vincoli (*constraints*) per determinare le configurazioni più appropriate di un sistema. Nel contesto di FREEDA, questa tecnica viene implementata attraverso MiniZinc [MiniZinc07], un linguaggio di modellazione che permette di esprimere problemi di ottimizzazione e di trovare la soluzione migliore rispettando le restrizioni definite. L'uso combinato di Continuous Reasoning e Constraint Reasoning consente a FREEDA di rispondere in modo efficace e tempestivo alle esigenze operative delle applicazioni distribuite.

Microservice-based Applications (MSA) in FREEDA Il paradigma (MSA) è al centro dell'architettura di FREEDA. Questo modello prevede la suddivisione delle applicazioni in servizi indipendenti (*microservizi*), ognuno con responsabilità ben definite e in grado di comunicare tra loro tramite API. L'adozione di MSA offre numerosi vantaggi, tra cui:

- **Scalabilità**: ogni microservizio può essere scalato indipendentemente dagli altri, consentendo un uso più efficiente delle risorse.
- **Manutenibilità**: grazie alla modularità dell'architettura, è possibile aggiornare, modificare o sostituire singoli componenti senza impattare sull'intero sistema.
- **Affidabilità**: la separazione in microservizi riduce il rischio che un guasto in un componente comprometta l'intero sistema.
- **Efficienza energetica**: la gestione dinamica delle risorse consente di allocare la potenza computazionale solo dove necessaria, minimizzando il consumo energetico.

In FREEDA, i microservizi vengono distribuiti sulle infrastrutture Cloud-IoT in modo intelligente, sfruttando le tecniche di *Continuous Reasoning* e *Constraint Reasoning* per adattarsi alle condizioni della rete e alle esigenze degli utenti in tempo reale. Questa combinazione di tecnologie rende il sistema più efficiente, resiliente e capace di gestire scenari complessi in maniera ottimizzata.

2.2 Tecnologie utilizzate

Nel progetto FREEDA sono state integrate diverse tecnologie fondamentali per garantire un'interfaccia utente avanzata, una gestione strutturata delle configurazioni e un'ottimizzazione delle risorse. Tra queste, JavaFX [JavaFX] e YAML giocano un ruolo chiave.

JavaFX è un framework per la creazione di interfacce grafiche moderne e interattive in Java. Offre un design flessibile e reattivo, con supporto nativo per CSS. In FREEDA, JavaFX è utilizzato per sviluppare l'interfaccia utente che permette agli utenti di definire e gestire le configurazioni del sistema in modo intuitivo, interagendo con i componenti del software tramite elementi grafici.

2.2.1 YAML

YAML (YAML Ain't Markup Language) è un formato di serializzazione dei dati pensato per essere leggibile dall'uomo e facilmente interpretato dalle macchine. La sua sintassi, basata su indentazione e coppie chiave-valore, lo rende uno strumento ideale per la configurazione di sistemi complessi e per lo scambio di dati tra applicazioni. Nel contesto di FREEDA, YAML viene impiegato per definire la configurazione dei microservizi e delle infrastrutture Cloud-IoT, facilitando la gestione delle dipendenze e delle risorse.

L'integrazione di queste tecnologie consente a FREEDA di offrire un ambiente configurabile e dinamico, migliorando l'usabilità e la scalabilità del sistema. La struttura minimalista di YAML permette di rappresentare dati complessi in modo ordinato e intuitivo. Questa semplicità facilita non solo la scrittura e la modifica manuale dei file di configurazione, ma anche la loro interpretazione da parte di sistemi automatizzati, riducendo il rischio di errori e rendendo il formato altamente accessibile.

Applicazioni e utilizzo Nel progetto FREEDA, YAML viene impiegato per definire le configurazioni dei componenti e per rappresentare in maniera standardizzata le architetture dei sistemi distribuiti. La scelta di YAML consente di automatizzare il processo di configurazione tramite un generatore dedicato. Tale generatore, attivato tramite l'interfaccia grafica (mediante la pressione del pulsante 2.1 *Genera YAML*), raccoglie i dati inseriti dall'utente, li elabora secondo logiche predefinite e produce un file YAML pronto per ulteriori elaborazioni. Questo approccio automatizzato garantisce precisione, uniformità e una significativa riduzione degli errori tipici della configurazione manuale.

Introduzione e Motivazione Nel contesto del progetto, l'obiettivo principale era gestire e rappresentare architetture complesse di sistemi distribuiti in modo strutturato e leggibile. Per raggiungere tale scopo, è stato sviluppato un *generatore YAML* che funge da strumento chiave per definire i componenti architetturali e le loro interdipendenze in un formato standardizzato. Il generatore si attiva quando l'utente interagisce con l'interfaccia grafica: una volta premuto il bottone Figura 2.1 *Genera YAML*, la classe raccoglie e processa i dati necessari, producendo un file di configurazione che rispecchia fedelmente la struttura del sistema FREEDA. Questa soluzione consente di semplificare la gestione delle configurazioni, garantendo una maggiore coerenza e flessibilità nell'amministrazione delle risorse, oltre a ridurre notevolmente gli errori derivanti dall'inserimento manuale dei dati.

Figura 2.1: Bottone che genera codice YAML

2.2.2 JavaFX

JavaFX [JavaFX] è un set di pacchetti grafici che consente agli sviluppatori Java di progettare, creare, testare, fare debug e distribuire applicazioni client ricche che operano in modo coerente su diverse piattaforme.

- **Architettura moderna:** JavaFX utilizza un'architettura moderna che sfrutta l'accelerazione hardware per il rendering grafico. Una parte delle operazioni di conversione dei dati viene eseguita dalla GPU, che è progettata per gestire elaborazioni parallele in modo efficiente.

- **Flessibilità e potenza:** Con JavaFX, gli sviluppatori possono creare interfacce grafiche utilizzando componenti standard, personalizzati e persino elementi in 3D. Inoltre, offre supporto per l'integrazione di contenuti web e multimediali, inclusi audio e video.

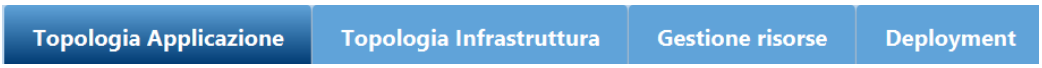
- **Binding e Proprietà:** JavaFX offre un avanzato sistema di binding che consente di collegare in modo intuitivo l'interfaccia utente alla logica dell'applicazione. Questo meccanismo garantisce che la UI si aggiorni automaticamente in base alle variazioni dello stato dell'applicazione, migliorando la reattività e riducendo la necessità di aggiornamenti manuali.

- **FXML:** JavaFX supporta FXML, un linguaggio di markup basato su XML, che consente di definire l'interfaccia utente in modo dichiarativo. In questa tesi, i file FXML sono stati utilizzati per definire la struttura delle varie schermate del pannello grafico. Questo approccio ha permesso di separare chiaramente la parte visiva dalla logica applicativa, migliorando la pulizia del codice e facilitando la manutenzione e l'aggiornamento dell'interfaccia, un esempio di configurazione si può vedere in Figura 2.2 *`Layout principale`* dove è stato implementato

il menù di navigazione principale con cui è possibile muoversi tra i vari layout.

```
1        <?xml version="1.0" encoding="UTF-8"?>
2        <?import javafx.scene.control.*?>
3        <?import javafx.scene.layout.*?>
4
5        <AnchorPane xmlns:fx="http://javafx.com/fxml"
    fx:controller="org.progettotesi.controller.MainController"
    >
6        <stylesheets>
7          <URL value="@css/FREEDA_style.css"/>
8        </stylesheets>
9          <TabPane fx:id="tabPane" AnchorPane.topAnchor="0"
    AnchorPane.bottomAnchor="0"
10       AnchorPane.leftAnchor="0" AnchorPane.rightAnchor="0">
11       <tabs>
12         <Tab fx:id="tabUses" text="Topologia Applicazione"
    closable="false"/>
13         <Tab fx:id="tabInfrastructure" text="Topologia
    Infrastruttura" closable="false"/>
14         <Tab fx:id="tabResources" text="Gestione risorse"
    closable="false"/>
15         <Tab fx:id="tabDeploy" text="Deployment" closable="
    false"/>
16       </tabs>
17         </TabPane>
18       </AnchorPane>
19
```

(a) Codice FXML



(b) Output grafico del codice

Figura 2.2: Layout principale

2.2.3 Minizinc

MiniZinc[MiniZinc07] è un linguaggio di modellazione ad alto livello per problemi di programmazione con vincoli (CSP - Constraint Satisfaction Problems). Viene utilizzato per descrivere problemi di ottimizzazione e di vincoli in modo dichiarativo, permettendo di separarli dalla strategia di risoluzione. MiniZinc è compatibile con diversi *solver* e consente di formulare problemi complessi in modo chiaro e strutturato.

Nel progetto FREEDA, MiniZinc è stato impiegato per elaborare e validare le configurazioni generate. Dopo la generazione dei file YAML temporanei, questi vengono passati ad un parser che genera la parte parametrica di un modello MiniZinc. Successivamente i solver trovano una soluzione ottima al problema. La soluzione viene poi passata successivamente all'interfaccia grafica per generare la topologia del deploy.

2.2.4 Maven

Maven [Maven] è un potente strumento di gestione dei progetti e automatizzazione dei processi di build utilizzato principalmente per progetti Java. Maven semplifica la gestione delle dipendenze, la compilazione, il test e la distribuzione del software, permettendo agli sviluppatori di concentrarsi sullo sviluppo delle funzionalità piuttosto che sulla gestione delle configurazioni. Una delle sue caratteristiche principali è la capacità di gestire le dipendenze tra librerie in modo automatico, evitando conflitti tra versioni e semplificando l'integrazione di nuove librerie.

Maven utilizza un file di configurazione XML, denominato *pom.xml* (Project Object Model), che definisce le dipendenze, i plugin, i repository e le configurazioni di build del progetto. Con una struttura di directory standardizzata, Maven facilita la gestione e la distribuzione dei progetti, supportando

anche la creazione di pacchetti, come file JAR (Java archive) o WAR (Web application archive), per il rilascio del software.

Nel progetto FREEDA, Maven è stato utilizzato per gestire in modo efficiente le librerie e le dipendenze necessarie per il funzionamento dell'applicazione. Grazie alla sua natura dichiarativa, Maven ha permesso di semplificare l'intero ciclo di vita del progetto, riducendo la complessità e migliorando l'affidabilità del processo di build e distribuzione.

Capitolo 3

Implementazione

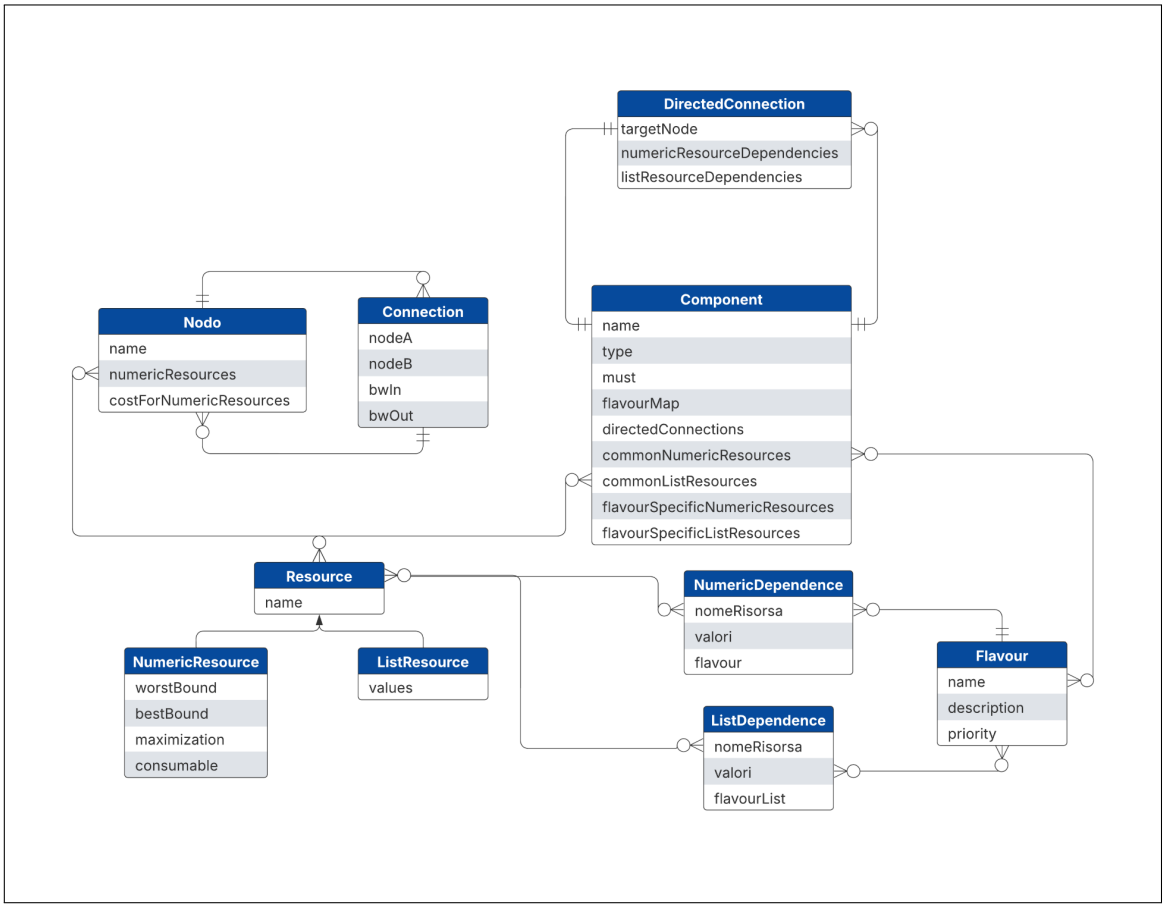


Figura 3.1: Diagramma ER del progetto

All'interno del progetto, sono state implementate diverse classi per riuscire a gestire tutte le possibili combinazioni. Come detto nel capitolo precedente, il lavoro svolto da JavaFX [JavaFX] deve cooperare perfettamente con il sistema di configurazione basato su YAML e con il motore di gestione delle risorse.

Per garantire questa integrazione, JavaFX fornisce un'interfaccia grafica interattiva che permette agli utenti di definire le configurazioni in modo intuitivo, mentre YAML gestisce la persistenza e la leggibilità dei dati. L'interazione tra JavaFX e YAML è stata progettata per essere modulare e scalabile, garantendo un'alta flessibilità nel processo di configurazione e gestione delle risorse all'interno dell'infrastruttura Cloud-IoT. Nella Figura 3.1 è possibile vedere un diagramma ER di riferimento delle classi che verranno successivamente illustrate

3.1 Component

La classe `Component` rappresenta un elemento centrale dell'architettura, identificato da attributi quali `name`, `type` e un flag `must` che indica se il componente è obbligatorio. Ogni componente ha bisogno di due cose per funzionare: ha bisogno della presenza di altri componenti (detta dipendenza funzionale) e ha bisogno di richiedere un certo numero di risorse (detta dipendenza su risorse)

La classe `Connection` e il relativo sistema di associazioni tra oggetti di tipo `Flavour` sono stati sviluppati per definire le relazioni tra varianti e, in particolare, per determinare quella più restrittiva. L'implementazione si basa sull'impiego congiunto di mappe e liste, strumenti essenziali per garantire una gestione flessibile e scalabile delle risorse e delle dipendenze.

Le mappe sono state utilizzate per stabilire le associazioni dirette tra le varianti. In particolare, ogni oggetto `Flavour` viene associato a un insieme

di attributi o ad altre varianti correlate attraverso l'utilizzo di una struttura dati come la `HashMap`. In questo contesto, la chiave della mappa rappresenta una specifica variante, mentre il valore associato (che può essere anch'esso una mappa o una lista) contiene le informazioni necessarie per determinare le restrizioni applicabili. Questo schema permette di effettuare ricerche rapide e di identificare con precisione la variante che presenta la configurazione più quella meno importante, garantendo così un'accurata gestione delle dipendenze.

Parallelamente, le liste sono impiegate per mantenere l'ordine di inserimento delle risorse e per gestire in maniera sequenziale i processi di validazione. Durante l'aggiunta di nuovi elementi, vengono applicati controlli sui valori nulli, assicurando che ogni risorsa inserita soddisfi i criteri previsti come il nome del componente e la presenza di almeno un flavour. Tale strategia riduce significativamente il rischio di errori, poiché ogni elemento viene verificato prima di essere integrato nella struttura dati complessiva.

3.2 Connection

La classe `Connection` rappresenta una connessione bidirezionale tra due nodi (di tipo `Nodo` spiegato successivamente). È progettata per memorizzare le informazioni relative a latenza e disponibilità, dati fondamentali per modellare le prestazioni e l'affidabilità della connessione. Tali parametri permettono di valutare in maniera accurata la qualità del collegamento, fornendo una base solida per l'ottimizzazione delle risorse e per la gestione efficace della rete.

3.3 DirectedConnection

La classe `DirectedConnection` definisce una connessione unidirezionale da un componente verso un altro, identificato come `targetNode`.

La classe `DirectedConnection` è stata sviluppata per gestire in modo efficace le dipendenze specifiche associate alle risorse, come `NumericDependence` e `ListDependence`. Questo meccanismo permette di monitorare e mantenere la coerenza della configurazione, garantendo che ogni relazione di dipendenza tra i componenti venga gestita in maniera accurata. Inoltre, sono stati implementati metodi dedicati alla rimozione delle dipendenze, operazione fondamentale per assicurare che eventuali modifiche o aggiornamenti non compromettano l'integrità dell'intero sistema. Come rappresentato nella Figura 3.2

```
1  public class DirectedConnection {
2      private Component targetNode;
3      private List<NumericDependence>
       numericResourceDependencies = new ArrayList<
       NumericDependence>();
4      private List<ListDependence> listResourceDependencies =
       new ArrayList<ListDependence>();
5
6      public DirectedConnection(Component targetNode) {
7          this.targetNode = targetNode;
8      }
9      public void removeNumericDependence(NumericDependence
       dependence) {
10         numericResourceDependencies.remove(dependence);
11     }
12     public void removeListDependence(ListDependence
       dependence) {
13         listResourceDependencies.remove(dependence);
14     }
15 }
```

Figura 3.2: Classe DirectedConnection.java

3.4 Flavour

La classe `Flavour` rappresenta una variante o configurazione specifica di un componente.

La classe `Flavour` si distingue per una serie di attributi fondamentali che ne definiscono l'identità e il comportamento all'interno della configurazione del sistema. In particolare:

- L'attributo `name` (di tipo stringa) identifica il nome del flavour, permettendo una facile distinzione tra le diverse varianti.
- L'attributo `description` fornisce una descrizione dettagliata, evidenziando le peculiarità e le specifiche del flavour.
- L'attributo `priority` (valore intero) stabilisce il livello di importanza del flavour, consentendo di determinare una gerarchia tra le varianti.

3.5 ListDependence

La classe `ListDependence` modella una dipendenza relativa a risorse di tipo lista, fornendo un meccanismo strutturato per gestire e organizzare le informazioni correlate a tali risorse.

- L'attributo `nomeRisorsa` contiene il nome identificativo della risorsa.
- L'attributo `valori` rappresenta una lista di valori correlati alla risorsa, offrendo flessibilità nella gestione di dati aggiornabili.
- L'attributo `flavourList` memorizza una collezione di oggetti di tipo `Flavour`, ciascuno definente una specifica configurazione o variante della risorsa.

Questa struttura consente di associare in modo preciso i vari valori a una particolare configurazione, permettendo di modellare accuratamente le dipendenze e le relazioni tra le risorse.

3.6 Resource

Essendo la classe base da cui derivano altre tipologie di risorse (come `NumericResource` e `ListResource`), garantisce una struttura comune per la gestione e l'estensione delle risorse, semplificando l'integrazione e la manutenzione del sistema.

3.7 ListResource

La classe `ListResource` estende la classe `Resource` per gestire risorse composte da una lista di stringhe.

- Fornisce metodi per aggiungere valori alla lista, garantendo che non vengano inseriti valori nulli o vuoti.
- Implementa metodi come `toString`, `equals` e `hashCode` per una corretta gestione e confronto delle risorse.

L'utilizzo di una struttura basata su liste permette di gestire insiemi di dati variabili in lunghezza e contenuto, offrendo la flessibilità necessaria per rappresentare configurazioni complesse in modo chiaro e coerente.

3.8 Nodo

La classe `Nodo` rappresenta un nodo dell'infrastruttura. Ogni nodo possiede:

- Un nome.

- Risorse numeriche, memorizzate in una mappa (`numericResources`), con un costo associato per ciascuna risorsa (`costForNumericResources`).
- Risorse di tipo lista, gestite tramite una lista di oggetti `ListResource`, con relativi costi per ciascun valore, organizzati in una mappa (`costForListValues`).
- Un carbon footprint, indicato tramite un valore intero, che rappresenta l'impatto ambientale.
- Un elenco di connessioni verso altri nodi, memorizzato in una lista (`connections`).
- Metodi per aggiungere risorse numeriche (`addNumericResource`) e per impostare il costo associato a ciascuna risorsa numerica (`setCostForResource`).
- Metodi per aggiungere risorse di tipo lista (`addListResource`) e per definire o aggiornare il costo di specifici valori all'interno di tali risorse (`setCostForListValue` e `updateCostForListValueByName`).
- Metodi per gestire le connessioni con altri nodi, consentendo di aggiungere (`addConnection`) o rimuovere (`removeConnection`) tali connessioni.

L'obiettivo di questa classe è modellare in maniera dettagliata ogni entità dell'infrastruttura, fornendo una base per l'allocazione delle risorse, la gestione dei costi e l'analisi dell'impatto ambientale tramite il carbon footprint.

3.9 NumericDependence

La classe `NumericDependence` modella una dipendenza numerica per una risorsa. Essa contiene:

- Il nome della risorsa (`nomeRisorsa`).
- Il valore numerico associato (`valore`).
- Un oggetto `Flavour` che specifica la variante o configurazione applicata alla dipendenza.

Questo modello facilita la gestione delle dipendenze tra le risorse numeriche, tracciando l'influenza di specifici `Flavour` nelle assegnazioni dei valori.

3.10 NumericResource

La classe `NumericResource` estende la classe astratta `Resource` e rappresenta risorse caratterizzate da proprietà numeriche. Gli attributi specifici includono:

- **worstBound** e **bestBound**: definiscono, rispettivamente upper e lower bound.
- **maximization**: un booleano che indica se la risorsa deve essere massimizzata.
- **consumable**: un flag che determina se la risorsa è consumabile.

Questa classe è fondamentale per la modellazione delle risorse numeriche, fornendo i parametri necessari per la gestione del loro consumo.

Capitolo 4

Configurazione YAML

In questo capitolo viene illustrato in dettaglio come le classi dell'applicazione lavorino per produrre file YAML standardizzati. Le classi come `Component`, `Nodo`, `NumericResource`, `ListResource`, `DirectedConnection`, `NumericDependence` e altre, cooperano all'interno della classe `YamlGenerator` per tradurre la complessità della rappresentazione interna in una struttura testuale facilmente leggibile e integrabile in altri sistemi.

La classe `Component` rappresenta l'unità fondamentale dell'applicazione, definendo i singoli elementi costitutivi del sistema. Ogni componente possiede attributi e metodi che ne permettono l'identificazione e la configurazione, e può essere arricchito con risorse specifiche. In questo contesto, le classi `NumericResource` e `ListResource` gestiscono rispettivamente dati quantitativi e collezioni di informazioni, garantendo una rappresentazione accurata e flessibile dei valori associati a ciascun componente.

La classe `Nodo` è responsabile dell'organizzazione logica e strutturale della rete dell'infrastruttura. Essa definisce i punti di collegamento tra i componenti, facilitando la creazione di relazioni e la trasmissione dei dati attraverso il sistema. Le connessioni tra questi nodi vengono ulteriormente modellate dalla classe `DirectedConnection`, che stabilisce relazioni unidirezionali, determinando il flusso informativo e mantenendo la coerenza delle

dipendenze.

Per quanto riguarda le dipendenze, classi come `NumericDependence` vengono impiegate per definire vincoli e relazioni specifiche tra le risorse. Queste dipendenze assicurano che, nel processo di configurazione, ogni relazione rispetti i criteri imposti dal sistema, garantendo integrità e coerenza nella struttura dati.

Al centro di questa architettura si trova la classe `YamlGenerator` svolge il ruolo di orchestratore. Essa raccoglie le informazioni provenienti dalle diverse classi, le elabora secondo regole predefinite e le trasforma in un file YAML. Durante il processo di serializzazione, vengono applicate regole di formattazione e validazione per assicurare che l'output finale sia conforme agli standard richiesti, facilmente leggibile e integrabile in altri sistemi. In questo modo, la complessità interna dell'applicazione viene tradotta in una rappresentazione testuale chiara e strutturata, facilitando la configurazione, l'analisi e l'interoperabilità.

Indice dei contenuti del capitolo Il capitolo si apre con un'analisi dettagliata del processo che porta alla generazione di file YAML per i componenti del sistema, illustrando come le diverse classi – quali `Component`,`Nodo`, `NumericResource`, `ListResource`, `DirectedConnection`, `NumericDependence` e altre – interagiscano tra loro. Ogni classe ha un ruolo specifico: ad esempio, le classi che gestiscono le risorse numeriche o le liste sono fondamentali per mantenere l'integrità e la coerenza della configurazione.

Al centro di questo processo troviamo il `YamlGenerator`, lo strumento che raccoglie e elabora i dati provenienti dai vari componenti, per poi serializzarli in un file YAML standardizzato. Questo meccanismo garantisce non solo un output coerente, ma anche una struttura facilmente leggibile e integrabile in altri sistemi.

Un ulteriore aspetto cruciale affrontato nel capitolo è la gestione delle ri-

sorse e delle dipendenze. Qui si approfondisce come i dati, siano essi numerici o strutturati in formato lista, vengono gestiti in modo tale da assicurare che le relazioni tra le diverse componenti mantengano la configurazione in uno stato valido e privo di errori. Le dipendenze vengono infatti strutturate per consentire un'interazione fluida e affidabile tra i moduli, rendendo il sistema robusto e scalabile.

Infine, viene esaminato l'output finale: il file YAML generato. Viene messa in luce la sua leggibilità, aspetto fondamentale per chi deve integrare e mantenere il sistema, e la facilità con cui questo file può essere utilizzato in ambienti di sviluppo e in sistemi di terze parti. Il formato YAML, infatti, favorisce una rapida comprensione e una gestione semplificata delle configurazioni, contribuendo a un'implementazione efficace e sicura.

4.1 Generazione YAML Componenti

Il metodo principale `generaYamlComponenti` prende in input una lista di nodi nodes e un percorso ad un file in formato *String* e si occupa di

- Creare una struttura radice (una mappa `root`) in cui viene impostato il nome principale dell'applicazione (ad es. `web_application`).
- Costruire la sezione `components`: per ogni oggetto `Component` presente nella lista, viene generata una sottosezione tramite il metodo `generaSezioneComponents(Component)`.
- La sezione `requirements` viene creata in due fasi complementari. Inizialmente, per ciascun componente del sistema viene generata la sua specifica sezione di requisiti, ottenuta iterando sui componenti e invocando il metodo `generaSezioneRequirements(Component)`. In questo modo, ogni componente contribuisce individualmente con i propri requisiti, garantendo una gestione modulare e dettagliata delle specifiche. Successivamente, viene costruita la sezione relativa alle dipendenze fra i componenti. Questa parte, realizzata tramite il metodo `generaSezioneDependencies(List<Component>)`, analizza le connessioni esistenti tra i componenti, considerando sia le dipendenze numeriche sia quelle strutturate come liste.
- Scrivere il file YAML risultante nel percorso specificato tramite un `FileWriter`.

La generazione del file YAML avviene costruendo sezioni specifiche per ogni componente e per i relativi requisiti. Di seguito, vengono illustrate le principali sezioni e i metodi che si occupano della loro creazione.

Sezione Components Il metodo `generaSezioneComponents(Component component)` si occupa della creazione della sezione `components` all'interno

del file YAML. L'obiettivo è tradurre le proprietà interne di un oggetto `Component` in una rappresentazione testuale strutturata e facilmente leggibile. In particolare, ogni componente viene rappresentato da una mappa che include le seguenti chiavi:

- **name**: il nome del componente, opportunamente convertito in minuscolo per garantire uniformità nell'output.
- **type**: il tipo del componente, che ne identifica la categoria o funzionalità all'interno del sistema.
- **must**: un valore booleano che indica se il componente è obbligatorio.
- **resources**: una struttura che organizza le risorse associate al componente, suddividendole in due sotto-mappe: una per le risorse numeriche e una per quelle di tipo lista.

Processo di generazione il metodo opera attraverso una serie di passaggi ben definiti:

1. **Estrazione e normalizzazione dei dati:** Il metodo accede alle proprietà del componente tramite i relativi metodi getter. In questa fase, il nome del componente viene convertito in minuscolo (utilizzando ad esempio il metodo `toLowerCase()`) per garantire una coerenza nell'output, mentre il tipo e il flag `must` vengono letti direttamente dall'oggetto.
2. **Organizzazione delle risorse:** dopo aver estratto le informazioni di base, il metodo procede a raccogliere le risorse associate al componente. Le risorse numeriche e quelle di tipo lista vengono separate in due mappe differenti, per una chiara distinzione fra le varie tipologie di dati

3. **Creazione della mappa finale:** le informazioni raccolte vengono aggregate in una mappa complessiva che rappresenta il componente. Tale mappa è poi passata a una libreria di serializzazione (come SnakeYAML), che si occupa di trasformarla in un file YAML strutturato e conforme agli standard richiesti.

Sezione Requirements Il metodo `generaSezioneRequirements(Component component)` crea i requisiti per ciascun componente. Ogni componente ha un insieme di risorse numeriche e di tipo lista che devono essere rispettate per garantirne il corretto funzionamento. Questa sezione include:

- **commonNumericResources**: risorse numeriche comuni definite come coppie chiave-valore.
- **commonListResources**: risorse di tipo lista con i relativi valori associati.
- **flavourSpecificNumericResources** e **flavourSpecificListResources**: risorse definite per specifici `Flavour` del componente.

Sezione Dependencies Il metodo `generaSezioneDependencies(List<Component> nodes)` analizza le connessioni tra i componenti per determinare le loro dipendenze. Questa sezione rappresenta:

- **Dipendenze dirette**: gestite attraverso la lista `directedConnections` di ciascun componente.
- **Dipendenze basate sulle risorse**: definite dalle relazioni tra risorse numeriche e di tipo lista tra diversi componenti.

Sezione Budget Il metodo `creaSezioneBudget()` definisce i costi e il carbon footprint dei componenti, che vengono inseriti al momento della creazione dei singoli componenti.

Per convertire la struttura dati Java in un file YAML leggibile, il metodo `convertToYaml(Object data, int level)` utilizza la libreria `SnakeYAML`. Questo metodo si occupa di trasformare oggetti Java in una rappresentazione YAML ben strutturata, garantendo che la formattazione rispetti la sintassi del linguaggio per eventuali altri sistemi che dovranno elaborare il file.

La conversione avviene ricorsivamente, un approccio essenziale per gestire correttamente strutture dati annidate e garantire una formattazione chiara. YAML si basa su un sistema di indentazione per rappresentare la gerarchia dei dati, quindi è fondamentale che ogni livello della struttura venga elaborato con la giusta profondità.

Per ottenere ciò, il metodo segue i seguenti principi:

- Se l'oggetto passato in input è una mappa (`Map<K, V>`), viene iterato chiave per chiave, assicurandosi che ciascun valore venga convertito ricorsivamente nel formato corretto.
- Se l'oggetto è una lista (`List<E>`), ogni elemento viene elaborato separatamente e formattato con il carattere speciale `-`.
- Se l'oggetto è una stringa, un numero o un valore booleano, viene semplicemente trasformato in una stringa testuale rispettando la sintassi YAML.

4.2 Generazione YAML Infrastrutture

Nome dell'Infrastruttura All'inizio del processo, viene definito il nome dell'infrastruttura. Il metodo aggiunge la seguente riga nel file YAML:

```
name: infrastructure
```

Questo identifica in modo univoco l'insieme dei nodi e dei collegamenti che compongono il sistema.

Sezione dei Nodi La sezione `nodes:` è generata iterando sulla lista dei nodi (`nodes`). Per ciascun nodo, il metodo esegue i seguenti passaggi:

1. **Identificazione del Nodo:** il nome del nodo viene utilizzato come chiave primaria all'interno della mappa, garantendo un'identificazione univoca.

2. Nel contesto della definizione delle **capacità di un nodo**, il metodo provvede a impostare, all'interno del file YAML, una sezione denominata capabilities. In questa sezione, il processo si articola in due fasi distinte.

   Per prima cosa, vengono gestite le risorse numeriche: il metodo itera su ciascuna coppia risorsa-valore, inserendola direttamente come una corrispondenza chiave-valore. In questo modo, ogni risorsa numerica viene espressa in maniera chiara e immediata, facilitando l'identificazione del relativo valore.

   Successivamente, per le risorse che presentano un formato a lista, il metodo raccoglie i valori corrispondenti e li raggruppa in una sintassi delimitata da parentesi quadre migliorando l'organizzazione dei dati.

3. **Profili di Costo:** ogni nodo include una sezione `profile` in cui vengono specificati:

   - **Cost:** una mappa che rappresenta i costi associati alle risorse numeriche. Il metodo itera sulle coppie risorsa-costo e formatta l'output in stile JSON, rimuovendo l'ultima virgola in eccesso.
   - **Carbon:** il valore del carbon footprint del nodo, indicato con la chiave `carbon`.

Sezione dei Collegamenti Successivamente, il metodo crea la sezione `links` che descrive le connessioni tra i nodi. Per ciascun collegamento presente nella lista (`connections`), vengono eseguiti i seguenti passaggi:

1. **Definizione dei Nodi Connessi:** viene generata la chiave `connected_nodes` con una lista contenente i nomi dei due nodi collegati.

2. **Capacità di un collegamento**: il sistema crea una mappa apposita in cui vengono specificati i parametri chiave per la gestione della connettività. In particolare, all'interno di questa mappa vengono indicati due aspetti fondamentali: la banda in ingresso, identificata con la chiave `bwIn`, e la banda in uscita, indicata con `bwOut`. Questi valori permettono di descrivere in modo dettagliato le prestazioni del collegamento, assicurando che il sistema possa valutare e gestire correttamente le risorse di rete disponibili.

Scrittura su File Al termine della generazione dell'intera struttura YAML, il contenuto viene scritto su file mediante un `FileWriter`. I passaggi principali includono:

- Apertura del file in scrittura Dall'argomento alla funzione `filePath`
- Scrittura dell'intero contenuto YAML, costruito precedentemente, all'interno del file.
- Gestione delle eccezioni tramite un blocco `try-catch` per intercettare eventuali errori I/O e sollevare un'eccezione dettagliata in caso di problemi durante la scrittura.

4.3 Generazione YAML Risorse

Il metodo `generaYamlResources(Resource resource)` si occupa di convertire un oggetto `Resource` in una mappa strutturata secondo il formato

YAML, facilitando così la serializzazione e la lettura delle proprietà della risorsa.

Il metodo segue questi passaggi principali:

1. **Creazione della Mappa:** viene creata una `LinkedHashMap` per mantenere l'ordine degli elementi durante la serializzazione.

2. **Verifica del Tipo di Risorsa:** utilizzando l'operatore `instanceof`, il metodo distingue tra le due tipologie di risorse:

   - **NumericResource:** Nel caso in cui la risorsa sia di tipo NumericResource, il sistema procede ad aggiungere una serie di chiavi per configurare il suo comportamento. In primo luogo, viene definita la chiave type, la quale assume il valore `consumable` se la risorsa è consumabile, oppure `non-consumable` in caso contrario. Successivamente, il parametro optimization viene impostato in base al risultato del flag isMaximization(), determinando se l'ottimizzazione avvenga in modalità `maximization` oppure `minimization`.
   - **ListResource:** Se la risorsa è di tipo `ListResource`, il metodo esegue due operazioni principali. Innanzitutto, aggiunge un elenco di valori ottenuti chiamando il metodo `listResource.getValues()`, che rappresentano le possibili scelte per quella risorsa. Successivamente, imposta il parametro **optimization** con il valore statico `"minimization"`, indicando che l'ottimizzazione sarà orientata alla minimizzazione.

4.4 Logica

Il generatore YAML segue una logica strutturata per tradurre la rappresentazione interna degli oggetti in un formato standardizzato e leggibile. Il

```
1     private static Map<String, Object> generaYamlResources(
      Resource resource) {
2         Map<String, Object> resourceMap = new LinkedHashMap
      <>();
3
4         if (resource instanceof NumericResource
      numericResource) {
5             resourceMap.put("type", numericResource.
      isConsumable() ? "consumable" : "non-consumable");
6             resourceMap.put("optimization", numericResource.
      isMaximization() ? "maximization" : "minimization");
7             resourceMap.put("worst_bound", numericResource.
      getWorstBound());
8             resourceMap.put("best_bound", numericResource.
      getBestBound());
9         } else if (resource instanceof ListResource
      listResource) {
10            resourceMap.put("choices", listResource.getValues
      ());
11            resourceMap.put("optimization", "minimization");
12        }
13
14        return resourceMap;
15    }
```

Codice 4.1: Codice per generare un file YAML delle risorse

processo si articola in diverse fasi, ciascuna delle quali è responsabile della trasformazione dei dati in una struttura YAML gerarchica.

4.4.1 Fase 1: Raccolta e Organizzazione dei Dati

La prima fase prevede la raccolta dei dati a partire dagli oggetti `Component` che compongono il sistema. Ogni componente possiede informazioni specifiche relative alle sue caratteristiche, alle risorse utilizzate e alle connessioni con altri componenti. Questi dati vengono strutturati in una serie di mappe nidificate per garantire un'organizzazione chiara e coerente. Prendiamo come esempio la `private Map<String, Map<String, String» deployMap = new HashMap<>();`, utilizza una struttura a due livelli: una chiave di tipo String viene associata a una mappa interna, anch'essa indicizzata da chiavi e valori di tipo String. Questa organizzazione consente di raggruppare le informazioni in una gerarchia logica, in cui la chiave esterna può rappresentare un componente o un modulo, mentre la mappa interna contiene le proprietà o configurazioni correlate a quel componente.

4.4.2 Fase 2: Costruzione della Struttura YAML

Una volta raccolti i dati, il generatore costruisce una struttura gerarchica organizzata in sezioni principali:

- **components**: rappresenta tutti i componenti del sistema con le rispettive caratteristiche.
- **requirements**: descrive i vincoli e le esigenze di ogni componente, come le risorse necessarie e le dipendenze.
- **dependencies**: mappa le relazioni tra i componenti, identificando eventuali collegamenti obbligati.
- **budget**: definisce i vincoli di costo e impatto ambientale.

4.4.3 Fase 3: Conversione e Formattazione

Una volta costruita la struttura dati, essa viene convertita in una stringa YAML mediante il metodo `convertToYaml()`. Questo metodo utilizza una logica ricorsiva per gestire la formattazione corretta:

- Se il valore è una lista di elementi semplici (stringhe, numeri), viene formattata in linea per migliorare la leggibilità.
- Se il valore è una mappa annidata, viene indentato correttamente per mantenere la struttura gerarchica.

4.4.4 Fase 4: Scrittura su File

Infine, il contenuto generato viene scritto su un file specificato dall'utente. Se il file non è accessibile o si verifica un errore durante la scrittura, il sistema intercetta l'eccezione e notifica l'utente con un messaggio di errore.

Grazie a questa logica ben definita, il generatore YAML garantisce coerenza nella rappresentazione dei dati, facilita l'integrazione con altri sistemi e riduce il rischio di errori dovuti alla gestione manuale della configurazione.

4.5 Fase di Deploy

La parte relativa alla generazione dei codici YAML e alla successiva elaborazione costituisce uno degli elementi più innovativi del progetto. In particolare, il sistema adotta un approccio articolato che prevede diverse fasi, ciascuna fondamentale per garantire una configurazione automatizzata e affidabile.

Innanzitutto, i dati relativi ai componenti e all'infrastruttura vengono trasformati in file YAML tramite l'utilizzo della classe dedicata, che attraverso il metodo `generaFileYamlTemp()` crea file temporanei in una directory appositamente scelta (la directory temporanea del sistema). Questo meccanismo non solo permette di isolare e gestire i file di configurazione in modo

sicuro, ma semplifica anche la fase di pulizia e aggiornamento dei dati, andando a sovrascrivere i dati ogni qualvolta viene avviato il deploy, così facendo ad ogni cambiamento che verrà effettuato questi dati verranno sovrascritti direttamente.

Successivamente, il sistema si occupa di preparare l'ambiente necessario per l'analisi e la validazione delle configurazioni. Per garantire che il parser Python sia sempre disponibile nella stessa cartella per tutti gli utenti, il nostro approccio prevede che una volta avviato il deploy, il codice del parser venga automaticamente inserito in una directory temporanea. In pratica, le funzioni `copyResourceFolder()` e `unzip()` copiano ed estraggono il contenuto dell'archivio compresso contenente il parser, posizionandolo nella cartella temporanea. In questo modo, non è necessaria alcuna configurazione manuale: il parser si trova sempre nello stesso percorso, pronto per essere eseguito. Successivamente, il parser viene avviato come processo esterno per analizzare i file YAML e generare un output strutturato in formato DZN, che funge da file di input per il solver. Infine, il file DZN viene passato a MiniZinc [MiniZinc07] attraverso il metodo `avviaMinizinc()`. Il solver, impiegando un modello predefinito, elabora l'input per produrre una configurazione ottimale che rispecchia le specifiche esigenze del sistema. Il risultato, organizzato in blocchi di output, viene quindi processato e reso disponibile per essere integrato nel flusso di lavoro complessivo.

Questa catena di elaborazione – dalla generazione dei file YAML, al parsing, fino al solver – non solo automatizza il processo di configurazione, riducendo significativamente il rischio di errori manuali.

Capitolo 5

Conclusioni

La presente tesi ha illustrato in maniera approfondita lo sviluppo di un sistema automatizzato per la generazione di file YAML, concepito per il progetto FREEDA. Attraverso l'integrazione di tecnologie come JavaFX [JavaFX], SnakeYAML [SnakeYAML] e MiniZinc, il lavoro si è concentrato sulla realizzazione di un'interfaccia utente intuitiva e di un motore di configurazione in grado di tradurre la complessità delle infrastrutture Cloud-IoT e dei microservizi in una rappresentazione testuale strutturata e facilmente gestibile.

Un aspetto fondamentale del lavoro svolto è stato lo sviluppo di un sistema per la generazione automatica di file YAML relativi a componenti, infrastrutture e risorse. Questo modulo, da me sviluppato, permette di definire in modo chiaro e strutturato la configurazione dei vari elementi del sistema, riducendo gli errori manuali e ottimizzando il flusso di lavoro. Inoltre, l'integrazione con MiniZinc [MiniZinc07] ha consentito di implementare meccanismi avanzati di ottimizzazione e verifica della configurazione, migliorando la gestione delle risorse e la distribuzione delle applicazioni.

Particolare attenzione è stata dedicata alla fase di deploy, garantendo che il sistema fosse in grado di generare configurazioni pronte per l'esecuzione in ambienti reali.

5.1 Lavori futuri

Per quanto riguarda i lavori futuri, una delle principali implementazioni previste riguarda il miglioramento dell'interfaccia grafica. Attualmente, l'aggiornamento delle configurazioni avviene in modo asincrono, ma un'estensione della GUI potrebbe permettere un aggiornamento istantaneo dell'interfaccia ogni volta che viene trovata una nuova soluzione da MiniZinc. Un aspetto particolarmente interessante riguarda l'aggiornamento automatico dell'interfaccia grafica: una volta che il solver produce una nuova soluzione, il sistema potrebbe aggiornare immediatamente la visualizzazione, offrendo così un feedback in tempo reale all'utente. Tale funzionalità rappresenterebbe un passo significativo per migliorare l'esperienza utente, garantendo una maggiore interattività e rendendo il processo di configurazione più dinamico ed efficace.

Inoltre, si potrebbe esplorare l'integrazione con sistemi di machine learning per migliorare ulteriormente l'efficienza nella generazione delle configurazioni, suggerendo automaticamente le migliori combinazioni sulla base di dati storici e preferenze dell'utente.

Bibliografia

[Maven] *Apache Maven - Welcome to Apache Maven*. `https://maven.apache.org/`.

[SnakeYAML] Andrey Asomov. *SnakeYAML - YAML parser for Java*. 2025. URL: `https://github.com/snakeyaml/snakeyaml`.

[JavaFX] *JavaFX Official Documentation*. URL: `https://openjfx.io/`.

[MiniZinc07] Nicholas Nethercote et al. "MiniZinc: Towards a Standard CP Modelling Language". In: *Principles and Practice of Constraint Programming – CP 2007*. A cura di Christian Bessière. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 529–543. ISBN: 978-3-540-74970-7.

[YAML] *YAML Official Website*. URL: `https://yaml.org/`.

Appendice A

File FXML

A.1 Main FXML

```
<AnchorPane xmlns:fx="http://javafx.com/fxml" fx:controller="
    org.progettotesi.controller.MainController">
     <stylesheets>
         <URL value="@css/FREEDA_style.css"/>
     </stylesheets>
     <TabPane fx:id="tabPane" AnchorPane.topAnchor="0"
    AnchorPane.bottomAnchor="0" AnchorPane.leftAnchor="0"
    AnchorPane.rightAnchor="0">
         <tabs>
             <Tab fx:id="tabUses" text="Topologia Applicazioni
    " closable="false"/>
             <Tab fx:id="tabInfrastructure" text="Topologia
    Infrastruttura" closable="false"/>
             <Tab fx:id="tabResources" text="Gestione risorse"
     closable="false"/>
             <Tab fx:id="tabDeploy" text="Deployment" closable
    ="false"/>
         </tabs>
     </TabPane>
</AnchorPane>
```

Codice A.1: File mainLayout.fxml

A.2 Pannello Applicazione

```
<AnchorPane xmlns:fx="http://javafx.com/fxml" fx:controller="
    org.progettotesi.controller.UsesPaneController">
     <stylesheets>
         <URL value="@css/FREEDA_style.css"/>
     </stylesheets>
     <children>
         <HBox alignment="CENTER"
               AnchorPane.topAnchor="14.0"
               AnchorPane.leftAnchor="0.0"
               AnchorPane.rightAnchor="0.0">
             <children>
                 <Label text="TOPOLOGIA APPLICAZIONE" style="-
    fx-font-size: 22px; -fx-font-weight: bold; -fx-font-
    family: 'Arial';"/>
             </children>
         </HBox>
         <VBox alignment="CENTER" spacing="10"
               AnchorPane.topAnchor="50.0"
               AnchorPane.leftAnchor="20.0"
               AnchorPane.rightAnchor="20.0">
             <children>
                 <HBox spacing="10" alignment="CENTER">
                     <children>
                         <Button fx:id="
    bottoneAggiungiComponente" text="Aggiungi Componente" />
                         <Button fx:id="
    bottoneRimuoviComponente" text="Rimuovi Componente" />
                         <Button fx:id="
    bottoneAggiungiDipendenza" text="Aggiungi Dipendenze" />
```

```
24                      <Button fx:id="bottoneAggiungiFlavour
    " text="Aggiungi Flavour" />
25                      <Button fx:id="bottoneGeneraYaml"
    text="Genera YAML" onAction="#handleGeneraYaml" />
26                  </children>
27              </HBox>
28              <Region fx:id="spacer" />
29          </children>
30      </VBox>
31      <Pane fx:id="pane" style="-fx-border-color: black; -
    fx-border-width: 2px;"
32            AnchorPane.topAnchor="100.0"
33            AnchorPane.leftAnchor="20.0"
34            AnchorPane.bottomAnchor="20.0"
35            AnchorPane.rightAnchor="450.0"
36            prefWidth="450"
37            prefHeight="300"/>
38      <VBox fx:id="dettagliComponentePane" visible="false"
39            style="-fx-border-color: black; -fx-padding:
    10;"
40            alignment="CENTER"
41            AnchorPane.topAnchor="100.0"
42            AnchorPane.rightAnchor="20.0"
43            AnchorPane.bottomAnchor="20.0"
44            prefWidth="400.0">
45          <children>
46              <Label fx:id="labelNomeComponente" style="-fx
    -font-size: 16px; -fx-font-weight: bold;" />
47              <VBox fx:id="flavoursList" spacing="5" />
48              <VBox fx:id="connectionsList" spacing="5" />
49              <HBox spacing="20" alignment="CENTER">
50                  <children>
51                      <Button fx:id="
    bottoneModificaComponente" text="Modifica" />
```

```
52                          <Button fx:id="
    bottoneChiudiDettaglioComponente" text="Chiudi" />
53                      </children>
54                  </HBox>
55              </children>
56          </VBox>
57      </children>
58  </AnchorPane>
```

Codice A.2: File useTopologyLayout.fxml

A.3 Pannello Infrastruttura

```
1  <?import java.net.URL?>
2  <AnchorPane xmlns:fx="http://javafx.com/fxml" fx:controller="
       org.progettotesi.controller.InfrastructurePaneController">
3      <stylesheets>
4          <URL value="@css/FREEDA_style.css"/>
5      </stylesheets>
6      <children>
7          <HBox alignment="CENTER"
8                AnchorPane.topAnchor="14.0"
9                AnchorPane.leftAnchor="0.0"
10               AnchorPane.rightAnchor="0.0">
11              <children>
12                  <Label text="TOPOLOGIA INFRASTRUTTURA"
13                         style="-fx-font-size: 22px; -fx-font-
       weight: bold; -fx-font-family: 'Arial';"/>
14              </children>
15         </HBox>
16         <VBox alignment="CENTER" spacing="10"
17               AnchorPane.topAnchor="50.0"
18               AnchorPane.leftAnchor="20.0"
19               AnchorPane.rightAnchor="20.0">
20              <children>
```

```
21                <HBox spacing="10" alignment="CENTER">
22                    <children>
23                        <Button fx:id="bottoneAggiungiNodo"
    text="Aggiungi Nodo" />
24                        <Button fx:id="bottoneRimuoviNodo"
    text="Rimuovi Nodo" />
25                        <Button fx:id="bottoneGeneraYaml"
    text="Genera YAML" />
26                    </children>
27                </HBox>
28                <Region fx:id="spacer" />
29            </children>
30        </VBox>
31        <Pane fx:id="pane" style="-fx-border-color: black; -
    fx-border-width: 2px;"
32              AnchorPane.topAnchor="100.0"
33              AnchorPane.leftAnchor="20.0"
34              AnchorPane.bottomAnchor="20.0"
35              AnchorPane.rightAnchor="450.0" />
36        <VBox fx:id="dettagliNodoPane" visible="false"
37              alignment="CENTER"
38              AnchorPane.topAnchor="100.0"
39              AnchorPane.rightAnchor="20.0"
40              AnchorPane.bottomAnchor="20.0"
41              prefWidth="400.0">
42            <children>
43                <Label fx:id="nomeNodoLabel" style="-fx-font-
    size: 16pt; -fx-font-weight: bold;" />
44                <VBox fx:id="connectionsList" spacing="5" />
45                <VBox fx:id="resourcesList" spacing="5" />
46                <HBox spacing="20" alignment="CENTER">
47                    <children>
48                        <Button fx:id="modificaNodoButton"
    text="Modifica" />
```

```
49                          <Button fx:id="
    chiudiDettagliNodoButton" text="Chiudi" />
50                      </children>
51                  </HBox>
52              </children>
53          </VBox>
54      </children>
55  </AnchorPane>
```

Codice A.3: File infrastructureTopologyLayout.fxml

A.4 Pannello Risorse

```
1  <?import java.lang.String?>
2  <AnchorPane xmlns="http://javafx.com/javafx"
3              xmlns:fx="http://javafx.com/fxml"
4              fx:controller="org.progettotesi.controller.
    ResourcesPaneController"
5              prefHeight="600.0" prefWidth="800.0">
6      <children>
7          <HBox alignment="CENTER"
8                AnchorPane.topAnchor="14.0"
9                AnchorPane.leftAnchor="0.0"
10               AnchorPane.rightAnchor="0.0">
11             <children>
12                 <Label text="GESTIONE RISORSE" style="-fx-
    font-size: 22px; -fx-font-weight: bold; -fx-font-family: '
    Arial';"/>
13             </children>
14         </HBox>
15         <VBox alignment="CENTER" spacing="10"
16               AnchorPane.topAnchor="50.0"
17               AnchorPane.leftAnchor="20.0"
18               AnchorPane.rightAnchor="20.0">
19             <children>
```

```
20                <HBox alignment="CENTER" spacing="10">
21                    <children>
22                        <Button fx:id="
    bottoneAggiungiRisorsaNumerica" text="Aggiungi risorsa
    numerica" />
23                        <Button fx:id="
    bottoneAggiungiRisorsaLista" text="Aggiungi risorsa lista"
     />
24                        <Button fx:id="generateYamlButton"
    text="Genera YAML Risorse" />
25                    </children>
26                </HBox>
27            </children>
28        </VBox>
29        <VBox fx:id="numericResourceConfig" spacing="10"
30              AnchorPane.topAnchor="20.0"
31              AnchorPane.leftAnchor="250.0"
32              visible="false">
33            <children>
34                <Label text="Configura Risorsa Numerica"
    style="-fx-font-size: 14px; -fx-font-weight: bold;" />
35                <HBox spacing="10">
36                    <children>
37                        <Label text="Nome:" />
38                        <TextField fx:id="
    numericResourceNameField" promptText="Inserisci il nome" /
    >
39                    </children>
40                </HBox>
41                <HBox spacing="10">
42                    <children>
43                        <Label text="Worst Bound:" />
44                        <TextField fx:id="
    numericWorstBoundField" promptText="Inserisci il Worst
    Bound" />
```

```
45                  </children>
46              </HBox>
47              <HBox spacing="10">
48                  <children>
49                      <Label text="Best Bound:" />
50                      <TextField fx:id="
    numericBestBoundField" promptText="Inserisci il Best Bound
    " />
51                  </children>
52              </HBox>
53              <HBox spacing="10">
54                  <children>
55                      <Label text="Tipo:" />
56                      <ComboBox fx:id="numericTypeComboBox"
    >
57                          <items>
58                              <FXCollections fx:factory="
    observableArrayList">
59                                  <String fx:value="
    Massimizzazione" />
60                                  <String fx:value="
    Minimizzazione" />
61                              </FXCollections>
62                          </items>
63                      </ComboBox>
64                  </children>
65              </HBox>
66              <HBox spacing="10">
67                  <children>
68                      <Label text="Consumable:" />
69                      <CheckBox fx:id="
    numericConsumableCheckBox" />
70                  </children>
71              </HBox>
```

```
72                <Button fx:id="saveNumericResourceButton"
    text="Salva Risorsa" />
73            </children>
74        </VBox>
75        <VBox fx:id="listResourceConfig" spacing="10"
76              AnchorPane.topAnchor="250.0"
77              AnchorPane.leftAnchor="250.0"
78              visible="false">
79            <children>
80                <Label text="Configura Risorsa di Tipo Lista"
     style="-fx-font-size: 14px; -fx-font-weight: bold;" />
81                <HBox spacing="10">
82                    <children>
83                        <Label text="Nome:" />
84                        <TextField fx:id="
    listResourceNameField" promptText="Inserisci il nome" />
85                    </children>
86                </HBox>
87                <Label text="Valori della Lista:" />
88                <VBox fx:id="listValuesContainer" spacing="5"
    >
89                    <children>
90                        <HBox spacing="10">
91                            <children>
92                                <TextField fx:id="
    newListValueField" promptText="Inserisci un valore" />
93                                <Button fx:id="
    addListValueButton" text="Aggiungi Valore" />
94                            </children>
95                        </HBox>
96                    </children>
97                </VBox>
98                <Button fx:id="saveListResourceButton" text="
    Salva Risorsa" />
99            </children>
```

```
100         </VBox>
101         <TableView fx:id="resourcesTable"
102                    AnchorPane.topAnchor="100.0"
103                    AnchorPane.leftAnchor="20.0"
104                    AnchorPane.rightAnchor="20.0"
105                    AnchorPane.bottomAnchor="20.0">
106             <columns>
107                 <TableColumn fx:id="resourceNameColumn" text=
    "Nome Risorsa" prefWidth="200" />
108                 <TableColumn fx:id="resourceTypeColumn" text=
    "Tipo" prefWidth="100" />
109                 <TableColumn fx:id="resourceDetailsColumn"
    text="Dettagli" prefWidth="460" />
110                 <TableColumn fx:id="resourceActionsColumn"
    text="Azioni" prefWidth="200" />
111             </columns>
112         </TableView>
113     </children>
114 </AnchorPane>
```

Codice A.4: File resourcesLayout.fxml

A.5 Pannello Deploy

```
<AnchorPane xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml" fx:controller="org.progettotesi.controller.DeployPaneController" prefHeight="400.0" prefWidth="600.0">
    <stylesheets>
        <URL value="@css/FREEDA_style.css"/>
    </stylesheets>
    <children>
        <!-- Header -->
        <HBox alignment="CENTER"
              AnchorPane.topAnchor="14.0"
              AnchorPane.leftAnchor="0.0"
              AnchorPane.rightAnchor="0.0">
            <children>
                <Label text="DEPLOYMENT" style="-fx-font-size: 22px; -fx-font-weight: bold; -fx-font-family: 'Arial';"/>
            </children>
        </HBox>
        <VBox alignment="CENTER" spacing="10"
              AnchorPane.topAnchor="50.0"
              AnchorPane.leftAnchor="20.0"
              AnchorPane.rightAnchor="20.0">
            <children>
                <HBox spacing="10" alignment="CENTER">
                    <children>
                        <Button fx:id="bottoneAvviaDeploy" text="Avvia Deploy"/>
                    </children>
                </HBox>
                <Region fx:id="spacer" />
            </children>
        </VBox>
```

```
28          <Pane fx:id="pane" style="-fx-border-color: black; -
    fx-border-width: 2px;"
29                AnchorPane.topAnchor="100.0"
30                AnchorPane.leftAnchor="20.0"
31                AnchorPane.bottomAnchor="20.0"
32                AnchorPane.rightAnchor="450.0" />
33          <HBox spacing="10" alignment="CENTER"
34                AnchorPane.topAnchor="70.0"
35                AnchorPane.rightAnchor="60.0">
36              <children>
37                  <VBox spacing="5" alignment="CENTER">
38                      <Label text="Flavour Priority"/>
39                      <ComboBox fx:id="comboFlavourPriority"
    value="incremental">
40                          <items>
41                              <FXCollections fx:factory="
    observableArrayList">
42                                  <String fx:value="incremental
    "/>
43                                  <String fx:value="manual"/>
44                                  <String fx:value="
    lexicographic"/>
45                                  <String fx:value="reversed"/>
46                              </FXCollections>
47                          </items>
48                      </ComboBox>
49                  </VBox>
50                  <VBox spacing="5" alignment="CENTER">
51                      <Label text="Solver"/>
52                      <ComboBox fx:id="comboSolver" value="
    chuffed">
53                          <items>
54                              <FXCollections fx:factory="
    observableArrayList">
55                                  <String fx:value="chuffed"/>
```

```
56                                  <String fx:value="gecode"/>
57                                  <String fx:value="highs"/>
58                                  <String fx:value="lcg"/>
59                              </FXCollections>
60                          </items>
61                      </ComboBox>
62                  </VBox>
63              </children>
64          </HBox>
65          <VBox fx:id="dettagliDeploy" visible="true"
66                style="-fx-border-color: black; -fx-padding:
    10;"
67                alignment="CENTER"
68                AnchorPane.topAnchor="250.0"
69                AnchorPane.rightAnchor="20.0"
70                AnchorPane.bottomAnchor="20.0"
71                prefWidth="400.0">
72              <children>
73                  <Label text="DEPLOY OUTPUT:" fx:id="
    deploymentOutputTitle" style="-fx-font-size: 20px; -fx-
    font-weight: bold;" />
74                  <VBox fx:id="deploymentOutput" spacing="5"
    style="-fx-padding: 10px 0 0 0;" />
75              </children>
76          </VBox>
77      </children>
78  </AnchorPane>
```

Codice A.5: File deployTopologyLayout.fxml

A.6 Genera YAML componenti

```
1  public static void generaYamlComponenti(List<Component> nodes
       , String filePath) {
2          Map<String, Object> root = new LinkedHashMap<>();
3          root.put("name", "app");
4
5          // 2. Sezione components
6          Map<String, Object> components = new LinkedHashMap
       <>();
7          for (Component component : nodes) {
8              components.put(component.getName().toLowerCase(),
        generaSezioneComponents(component));
9          }
10         root.put("components", components);
11
12         // 3. Sezione requirements
13         Map<String, Object> requirements = new LinkedHashMap
       <>();
14
15         // 3.1. requirements/components
16         Map<String, Object> componentsRequirements = new
       LinkedHashMap<>();
17         for (Component component : nodes) {
18             componentsRequirements.put(component.getName().
       toLowerCase(), generaSezioneRequirements(component));
19         }
20         requirements.put("components", componentsRequirements
       );
21
22         // 3.2. requirements/dependencies
23         requirements.put("dependencies",
       generaSezioneDependencies(nodes));
24
25         // 3.3. requirements/budget
```

```
26          requirements.put("budget", creaSezioneBudget());
27
28          root.put("requirements", requirements);
29
30          // Conversione in struttura (mappe) in struttura
    stringa
31          String yamlString = convertToYaml(root, 0);
32
33          // Scrittura su file
34          try (FileWriter writer = new FileWriter(filePath)) {
35              writer.write(yamlString);
36              System.out.println("File YAML generato con
    successo: " + filePath);
37          } catch (IOException e) {
38              e.printStackTrace();
39              throw new RuntimeException("Errore durante la
    scrittura del file YAML: " + filePath, e);
40          }
41      }
```

Codice A.6: generaYamlComponeneti.java

A.7 Genera YAML infrastrutture

```
1  public static void generaYamlInfrastruttura(List<Nodo> nodes,
       List<Connection> connections, String filePath) {
2      StringBuilder yamlOutput = new StringBuilder();
3      yamlOutput.append("name: infrastructure\n");
4      yamlOutput.append("nodes:\n");
5
6      for (Nodo node : nodes) {
7          yamlOutput.append("  ").append(node.getName()).append
       (":\n");
8          yamlOutput.append("    capabilities:\n");
9
10         node.getNumericResources().forEach((resource, value)
       -> {
11             yamlOutput.append("      ").append(resource.
       getName()).append(": ").append(value).append("\n");
12         });
13
14         for (ListResource listResource : node.
       getListResources()) {
15             yamlOutput.append("      ")
16                     .append(listResource.getName())
17                     .append(": [")
18                     .append(String.join(", ", listResource.
       getValues()))
19                     .append("]\n");
20         }
21
22         yamlOutput.append("    profile:\n");
23         yamlOutput.append("      cost: { ");
24
25         node.getCostForNumericResources().forEach((resource,
       cost) -> {
```

```
26              yamlOutput.append(resource.getName()).append(": "
    ).append(cost).append(", ");
27          });
28
29          yamlOutput.setLength(yamlOutput.length() - 2);
30          yamlOutput.append(" }\n");
31          yamlOutput.append("      carbon: ").append(node.
    getCarbon()).append("\n");
32      }
33      yamlOutput.append("links:\n");
34      for (Connection connection : connections) {
35          yamlOutput.append("  - connected_nodes: [ ")
36                  .append(connection.getNodeA().getName())
37                  .append(", ")
38                  .append(connection.getNodeB().getName())
39                  .append(" ]\n");
40          yamlOutput.append("    capabilities: { 'bwIn': ")
41                  .append(connection.getBwIn())
42                  .append(", 'bwOut': ")
43                  .append(connection.getBwOut())
44                  .append(" }\n");
45      }
46      try (FileWriter writer = new FileWriter(filePath)) {
47          writer.write(yamlOutput.toString());
48          System.out.println("File YAML generato con successo:
    " + filePath);
49      } catch (IOException e) {
50          e.printStackTrace();
51          throw new RuntimeException("Errore durante la
    scrittura del file YAML: " + filePath, e);
52      }
53  }
```

Codice A.7: Codice per generare un file YAML delle infrastrutture

Appendice B

Esempi file YAML generati

B.1 Esempio file YAML dei componenti

```yaml
name: app
components:
  c0:
    type: service
    must: false
    flavours:
      f0:
        uses:
          - { component: c1, min_flavour: f0 }
      f1:
        uses:
          - { component: c1, min_flavour: f0 }
    importance_order: [f0, f1]
  c1:
    type: service
    must: true
    flavours:
      f0:
        uses:
          - { component: c2, min_flavour: f0 }
```

```
21      importance_order: [f0]
22    c2:
23      type: service
24      must: true
25      flavours:
26        f0:
27          uses:
28            - { component: c3, min_flavour: f0 }
29      importance_order: [f0]
30    c3:
31      type: service
32      must: false
33      flavours:
34        f0:
35          uses:
36            - { component: c4, min_flavour: f0 }
37      importance_order: [f0]
38    c4:
39      type: service
40      must: true
41      flavours:
42        f0:
43          uses:
44            - { component: c5, min_flavour: f0 }
45      importance_order: [f0]
46    c5:
47      type: service
48      must: false
49      flavours:
50        f0:
51          uses:
52            - { component: c6, min_flavour: f0 }
53      importance_order: [f0]
54    c6:
55      type: service
```

```
    must: false
    flavours:
      f0:
        uses: []
      f1:
        uses: []
      f2:
        uses:
          - { component: c7, min_flavour: f0 }
    importance_order: [f0, f1, f2]
  c7:
    type: service
    must: true
    flavours:
      f0:
        uses: []
      f1:
        uses: []
    importance_order: [f0, f1]

requirements:
  components:
    c0:
      common:
        storage: { value: 128 }
      flavour-specific:
        f0: { ram: { value: 67 } }
        f1: { ram: { value: 17 } }
    c1:
      common: { ram: { value: 133 } }
      flavour-specific:
        f0: { cpu: { value: 4 } }
    c2:
      common: {}
      flavour-specific:
```

```
91          f0: { ram: { value: 25 } }
92      c3:
93        common: {}
94        flavour-specific:
95          f0: { ram: { value: 11 } }
96      c4:
97        common: {}
98        flavour-specific:
99          f0: { ram: { value: 61 } }
100     c5:
101       common: { storage: { value: 121 } }
102       flavour-specific:
103         f0: { ram: { value: 58 } }
104     c6:
105       common: { ram: { value: 104 } }
106       flavour-specific:
107         f0: { cpu: { value: 4 } }
108         f1: { cpu: { value: 4 } }
109         f2: { cpu: { value: 3 } }
110     c7:
111       common: {}
112       flavour-specific:
113         f0: { ram: { value: 74 } }
114         f1: { ram: { value: 28 } }
115
116   dependencies:
117     c0:
118       f1:
119         c1: { bwIn: { value: 102 }, bwOut: { value: 98 } }
120     c1:
121       f0:
122         c2: { bwIn: { value: 17 }, bwOut: { value: 42 } }
123     c2:
124       f0:
125         c3: { bwIn: { value: 37 }, bwOut: { value: 64 } }
```

```
126     c3:
127       f0:
128         c4: { bwIn: { value: 104 }, bwOut: { value: 84 } }
129     c4:
130       f0:
131         c5: { bwIn: { value: 77 }, bwOut: { value: 27 } }
132     c5:
133       f0:
134         c6: { bwIn: { value: 9 }, bwOut: { value: 73 } }
135     c6:
136       f2:
137         c7: { bwIn: { value: 89 }, bwOut: { value: 66 } }
138
139   budget:
140     cost: 2000000
141     carbon: 2000000
```

Codice B.1: File components.yaml

B.2 Esempio file YAML dell'Infrastruttura

```
1  name: infrastructure
2  nodes:
3    node_0:
4      capabilities:
5        cpu: 674
6        ram: 745
7        storage: 755
8      profile:
9        cost: { cpu: 1, ram: 1, storage: 1 }
10       carbon: 9
11   node_1:
12     capabilities:
13       cpu: 127
14       ram: 178
```

```yaml
      storage: 580
    profile:
      cost: { cpu: 1, ram: 1, storage: 1 }
      carbon: 6
  node_2:
    capabilities:
      cpu: 195
      ram: 245
      storage: 178
    profile:
      cost: { cpu: 1, ram: 1, storage: 1 }
      carbon: 3
  node_3:
    capabilities:
      cpu: 134
      ram: 348
      storage: 580
    profile:
      cost: { cpu: 1, ram: 1, storage: 1 }
      carbon: 8
  node_4:
    capabilities:
      cpu: 672
      ram: 328
      storage: 671
    profile:
      cost: { cpu: 1, ram: 1, storage: 1 }
      carbon: 2
  node_5:
    capabilities:
      cpu: 882
      ram: 670
      storage: 542
    profile:
      cost: { cpu: 1, ram: 1, storage: 1 }
```

```yaml
      carbon: 7
  node_6:
    capabilities:
      cpu: 813
      ram: 722
      storage: 774
    profile:
      cost: { cpu: 1, ram: 1, storage: 1 }
      carbon: 6
links:
  - connected_nodes: [ node_0, node_1 ]
    capabilities: { bwIn: 585, bwOut: 303 }
  - connected_nodes: [ node_0, node_2 ]
    capabilities: { bwIn: 932, bwOut: 491 }
  - connected_nodes: [ node_0, node_3 ]
    capabilities: { bwIn: 367, bwOut: 431 }
  - connected_nodes: [ node_0, node_4 ]
    capabilities: { bwIn: 960, bwOut: 694 }
  - connected_nodes: [ node_0, node_5 ]
    capabilities: { bwIn: 113, bwOut: 850 }
  - connected_nodes: [ node_0, node_6 ]
    capabilities: { bwIn: 150, bwOut: 97 }
  - connected_nodes: [ node_2, node_6 ]
    capabilities: { bwIn: 907, bwOut: 457 }
  - connected_nodes: [ node_3, node_6 ]
    capabilities: { bwIn: 868, bwOut: 876 }
  - connected_nodes: [ node_4, node_6 ]
    capabilities: { bwIn: 377, bwOut: 136 }
  - connected_nodes: [ node_5, node_6 ]
    capabilities: { bwIn: 347, bwOut: 802 }
```

Codice B.2: File infrastructure.yaml

B.3 Esempio file YAML delle risorse

```yaml
cpu:
  type: consumable
  optimization: minimization
  worst_bound: 0.0
  best_bound: 10.0
ram:
  type: consumable
  optimization: minimization
  worst_bound: 0.0
  best_bound: 10.0
storage:
  type: consumable
  optimization: minimization
  worst_bound: 0.0
  best_bound: 10.0
bwIn:
  type: consumable
  optimization: minimization
  worst_bound: 0.0
  best_bound: 10.0
bwOut:
  type: consumable
  optimization: minimization
  worst_bound: 0.0
  best_bound: 10.0
availability:
  type: consumable
  optimization: minimization
  worst_bound: 0.0
  best_bound: 10.0
latency:
  type: consumable
  optimization: minimization
```

```
34    worst_bound: 0.0
35    best_bound: 0.0
36  security:
37    choices:
38    - ssl
39    - firewall
40    - encrypted_storage
41    optimization: minimization
```

Codice B.3: File resources.yaml