POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Informatica

Tesi di Laurea Magistrale

Cloud Computing for the masses: A Kubernetes-based Desktop Environment

Relatore
Prof. Fulvio RISSO

Candidato
Pietro Claudio LAGRECA

Supervisore
Dott. Marco IORIO

Anno Accademico 2021-2022

Sommario

I servizi di cloud computing stanno diventando ogni giorno più diffusi, grazie agli elevati vantaggi che essi offrono, alcuni su tutti la flessibilità delle risorse e l'assenza dei costi di gestione dell'hardware. Tuttavia, la creazione e la manutenzione di tali servizi non sono banali; inoltre, gli utenti si affidano molto a fornitori di cloud pubblici (ad esempio Google, Amazon, Oracle) che offrono servizi già pronti. Ciò è controproducente per gli utenti, perché non sono invogliati ad esplorare e utilizzare il mondo cloud.
Questo progetto di tesi è incentrato sul contesto di CrownLabs, un fornitore di servizi cloud basato su Kubernetes, e reso disponibilie da un gruppo di ricerca del Politecnico di Torino durante i periodi di chiusura dell'Ateneo per la pandemia da Coronavirus.
Il progetto mira ad esplorare l'idea di CrownLabs di fornire servizi cloud a un bacino d'utenza più ampio rispetto ai soli studenti del Politecnico aggiungendo la capacità di supportare servizi arbitrari e la possibilità di avviare applicazioni come su un computer desktop tradizionale.
In aggiunta a ciò, altro obiettivo del progetto è stato quello di fornire un meccanismo di contabilità per tenere sotto controllo l'utilizzo delle risorse da cui scaturisce la possibilità di personalizzare le risorse utilizzate da ciascun servizio (ad esempio, assegnare il valore corretto di CPU) e da ciascun utente (ogni utente ha un numero limitato di risorse a disposizione, ma il limite non deve essere fisso, bensì calcolato in base al suo consumo e alle sue necessità).
Il primo caso d'uso trattato nel lavoro a questo progetto consiste nell'integrazione in CrownLabs di Faveo Helpdesk, un software open source di assistenza basata sul ticketing. Questo consente a tutti gli utenti di CrownLabs di richiedere supporto senza necessità di avere recapiti diretti di chi dovrà fornire assistenza, e inoltre garantisce una migliore distribuzione del flusso di richieste sfruttando la caratteristica del ticketing di suddividere le comunicazioni in macro aree di competenza.
Il successivo caso d'uso trattato consiste nell'ottimizzazione della gestione della quantità di risorse computazionali assegnate a ciascun utente.

Indice

Elenco delle figure

Lista degli Acronimi

Capitolo 1

Introduzione generale

1.1 Cosa sono i Servizi Cloud

I servizi cloud [14] sono costituiti da infrastrutture, piattaforme o software in hosting presso provider esterni e messi a disposizione degli utenti attraverso Internet. Esistono molte tipologie di servizi cloud, tutte accomunate dalla caratteristica di non richiedere download di software aggiuntivo. In questo elaborato il focus è sulle tipologie elencate di seguito:

- **Infrastructure-as-a-Service (IaaS)**, si offrono all'utente risorse per permettere lo storage in memoria, risorse di rete e di elaborazione
- **Platforms-as-a-Service (PaaS)**, si offre agli utenti una piattaforma completa per l'esecuzione delle applicazioni, oltre a tutta l'infrastruttura IT necessaria
- **Software-as-a-Service (SaaS)**, si rende disponibile per l'utente un'intera applicazione cloud, completa di piattaforma di esecuzione e infrastruttura per costruire la piattaforma

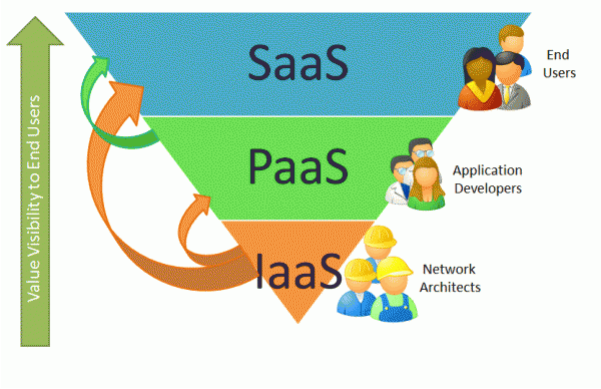

Figura 1.1: Tipologie di servizi Cloud

Al pari di tutte le soluzioni IT i servizi cloud sono costituiti da hardware e software, esiste però una grande differenza che consiste nel fatto che per poter sfruttare questi servizi gli utenti necessitano semplicemente di un personal computer, un sistema operativo e una connessione ad internet.
Tipicamente un'infrastruttura Cloud a livello hardware è caratterizzata dalla separazione delle funzionalità di elaborazione dai componenti fisici, quali RAM, CPU e data center o dischi rigidi per lo storage.
Una piattaforma Cloud funzionante prevede inoltre che le funzionalità di elaborazione offerte dall'hardware collaborino tra loro tramite l'integrazione delle tecnologie di containerizzazione, orchestrazione, interfacce di programmazione delle applicazioni, routing e sicurezza.
Per quanto riguarda il software, esso viene offerto agli utenti in modalità simili a quelle di un'applicazione online, con l'approccio **Cloud Native**, cioè un'architettura che combina tra loro molteplici microservizi indipendenti. Questo tipo di soluzione offre il vantaggio di creare applicazioni reattive, scalabili, e in grado di tollerare gli errori.

1.2 Obiettivo

La presente attività di tesi è incentrata sulla scoperta del mondo dei servizi Cloud e sul lavoro svolto da coloro che li forniscono per permettere a chi ne usufruisce di concentrarsi solo sul loro effettivo utilizzo e non sulla manutenzione e i costi di gestione dell'hardware.
Attualmente i servizi Cloud utilizzano una grande quantità di risorse computazionali data la loro notevole diffusione (basti pensare al grande ecosistema di Google Drive). Tuttavia la maggior parte degli utenti, non essendo specializzata nel campo, non è sensibilizzata al meglio all'utilizzo di tali risorse nella maniera più

corretta supponendo erroneamente che queste stesse risorse siano illimitate. Da qui nasce l'esigenza di gestire al meglio la potenza di calcolo delle macchine sulle quali i servizi cloud sono esposti e di limitare gli stessi utenti nel consumo di questa potenza.
Il progetto di tesi ha un particolare focus su due principali aspetti:

- Il primo consiste nel fornire un esempio di come si può rendere un servizio disponibile agli utenti seguendo il paradigma **SaaS**. Il servizio in questione è **Faveo Helpdesk**, un software di assistenza basata su **Ticketing** che non segue l'approccio Cloud Native, da quì nasce l'esigenza di creare i microservizi necessari a fornire il supporto cloud e rendere il tutto funzionante.

- Il secondo aspetto esplora il paradigma **IaaS**, e consiste nel presentare un esempio specifico di regolarizzazione del consumo di risorse computazionali fornite agli utenti, in modo da evitare sovraccarichi di lavoro da parte di chi fornisce queste risorse e l'esaurimento delle risorse stesse.

.

I prossimi capitoli forniscono una panoramica del contesto in cui ha avuto luogo il lavoro svolto in questo progetto, introducendo la tecnologia di Kubernetes e il progetto CrownLabs, che cosituisce un esempio di piattaforma che segue il paradigma PaaS basato su Kubernetes.

Capitolo 2

Kubernetes

Questo capitolo introduce **Kubernetes** [11], un sistema open-source di orchestrazione e gestione di container sul quale sono focalizzati tutti gli argomenti trattati nelle attività di questo progetto di tesi.
Kubernetes è una piattaforma portatile ed estensibile sviluppata con il linguaggio Go [6] ed è stato ideato per la gestione di carichi di lavoro e servizi containerizzati [8], in grado di facilitare sia la configurazione dichiarativa che l'automazione.
L'utilizzo dei container rappresenta una valida soluzione per distribuire ed eseguire applicazioni in un contesto di produzione all'interno del quale è necessario garantire che non si verifichino interruzioni dei servizi.
Per esempio, se un container si interrompe, è necessario avviarne uno nuovo, in modo tale che l'utente non percepisca che si è verificato un guasto al sistema.
Questi aspetti sono tutti gestiti da Kubernetes, che fornisce un framework per far funzionare i sistemi distribuiti in modo resiliente.
In particolare Kubernetes fornisce:

- **Bilanciamento del carico di lavoro**, se Kubernetes rileva che il traffico verso un container è alto, si occupa di ridistribuirlo su più container in modo che il servizio rimanga stabile
- **Rollout e rollback automatizzati**, Kubernetes permette di descrivere lo stato desiderato per i propri container, e si occuperà di cambiare periodicamente lo stato attuale per raggiungere quello desiderato ad una velocità controllata
- **Ottimizzazione dei carichi**, quando si fornisce a Kubernetes un cluster di nodi per eseguire i container lo si può istruire su quanta CPU e memoria (RAM) ha bisogno ogni singolo container. Kubernetes allocherà i container sui nodi per massimizzare l'uso delle risorse a disposizione.
- **Failover**, capacità di sostituire un container non funzionante con uno analogo

Nelle prossime sezioni si illustrano alcuni componenti di Kubernetes che sono stati protagonisti delle attività svolte in questo progetto.

2.1 Cluster

Un cluster Kubernetes è un gruppo di nodi utile per eseguire applicazioni containerizzate, e rappresenta l'elemento centrale di Kubernetes. Un nodo è una macchina virtuale o fisica, a seconda della tipologia del cluster.
Un cluster a livello base contiene un nodo Master, chiamato anche **Control Plane** e uno o più nodi che usati come sistemi di elaborazione detti anche **Worker Node**.
Il Control Plane mantiene lo stato desiderato del cluster, decidendo ad esempio le applicazioni da eseguire e i container sui quali queste ultime sono contenute, mentre i Worker Nodes eseguono concretamente le applicazioni e i carichi di lavoro.
La definizione di stato desiderato di un'applicazione avviene tramite file di configurazione manifest, ovvero file di tipo JSON o YAML che dichiarano il tipo di applicazione da eseguire e il numero di repliche necessarie per eseguire un sistema preservandone l'integrità.
Le comunicazioni tra utente e cluster avvengono invece tramite la Kubernetes Application Programming Interface (API), che si occupa di stabilire se una richiesta è valida, prima di poter procedere con la sua elaborazione.

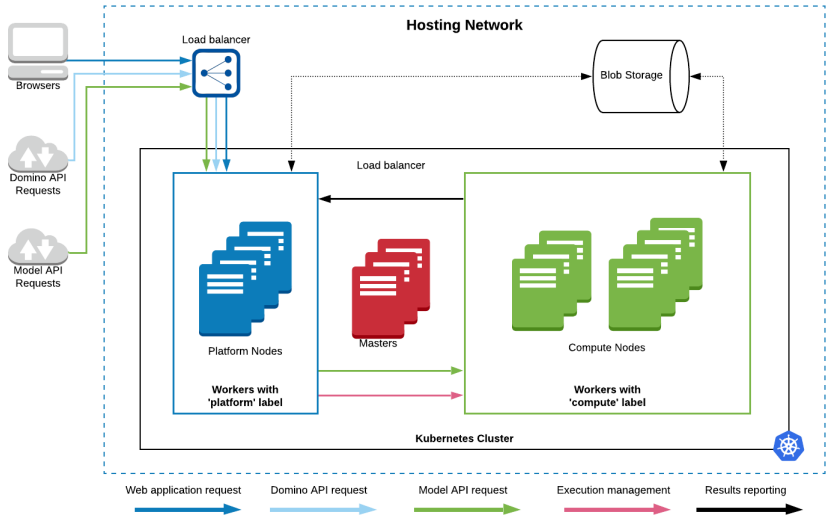

Figura 2.1: Architettura di un Cluster Kubernetes

2.2 Custom Resource Definition

Le Custom Resource Definition (CRD) sono oggetti che estendono le API di Kubernetes e definiscono un nuovo tipo di oggetto utilizzabile all'interno del cluster, anche esse vengono create a partire da un file manifest. Una volta definite le CRD gli oggetti che implementano il nuovo tipo sono chiamati Custom Resource (CR). Il seguente box propone esempio di file manifest scritto in YAML che definisce una CRD, in questo file sono inseriti i nomi e i tipi di tutti gli attributi di cui l'oggetto che implementa la CRD deve essere dotato.

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                cronSpec:
                  type: string
                image:
                  type: string
                replicas:
                  type: integer
  scope: Namespaced
  names:
    plural: crontabs
    singular: crontab
    kind: CronTab
    shortNames:
    - ct
```

Listing 2.1: Esempio di una CRD

Per rendere la CRD disponibile sul cluster a partire dal suo file manifest bisogna creare la risorsa Kubernetes usando il comando *kubectl apply* su terminale di riga di comando passando in input il manifest.
Tra tutti gli attributi degli oggetti in Kubernetes ce ne sono due che meritano un maggiore approfondimento:

- **Spec**, rappresenta lo stato desiderato dell'oggetto
- **Status**, rappresenta lo stato corrente dell'oggetto risorsa

È compito degli **operatori** fare in modo che lo stato desiderato venga rispecchiato in quello corrente, il loro funzionamento sarà descritto nelle prossime sezioni.

2.3 Namespace

In Kubernetes i Namespace forniscono un meccanismo per isolare gruppi di risorse all'interno di un cluster, perciò in un cluster con multipli namespace il nome deve essere univoco, così come i nomi delle risorse all'interno del singolo namespace. È possibile definire CRD applicabili solo all'interno dello scope di un namespace, come è stato fatto nell'esempio di manifest fornito nella sezione precedente.
All'atto della creazione di un cluster, Kubernetes fornisce quattro inziali namespace:

- **default**, il namespace che contiene tutti gli oggetti non contenuti in altri namespace.
- **kube-system**, contiene tutti gli oggetti di sistema.
- **kube-public**, namespace pubblico ideato per contenere oggetti accessibili a tutti gli utenti.
- **kube-node-lease**, contiene gli oggetti **Lease** per ogni nodo, utili per inviare segnali in modo che il Control Plane possa rilevare eventuali errori del nodo.

In un cluster esistono oggetti indipendenti dai namespace, ad esempio i nodi, e oggetti che possono essere creati solo all'interno di un namespace, ad esempio **Deployment** e **Service**, ciò rende necessaria la presenza dei namespace all'interno del cluster.

2.4 Container

Come suggerisce il nome questi oggetti sono veri e propri contenitori di applicazioni, essi forniscono quel complesso di dati che occorrono a un'applicazione cloud native per essere eseguita e permettono di disaccoppiare le applicazioni dal sistema operativo su cui esse sono eseguite.
Un container è costruito a partire da un'**immagine**, ossia un pacchetto software che contiene tutto l'occorrente per eseguire un'applicazione: librerie applicative e di sistema, le impostazioni predefinite per ogni configurazione necessaria e il codice sorgente. In particolare, quest'ultimo è fondamentale affinchè il container abbia

un corretto punto di partenza all'interno del processo di elaborazione che porta all'esecuzione dell'applicazione stessa.

2.5 Pod

Il pod è la più elementare unità di pianificazione dei container che può essere creata e gestita su Kubernetes. Come suggerisce il nome, che dall'inglese si traduce con "baccello", il pod è un gruppo di uno o più container con risorse di rete condivise, e contiene inoltre le specifiche su come i singoli container devono essere eseguiti.
I componenti descritti finora e la loro interazione caratterizzano il cluster Kubernetes e il suo comportamento, e identificano lo scheletro dei servizi messi a disposizione per l'utente.

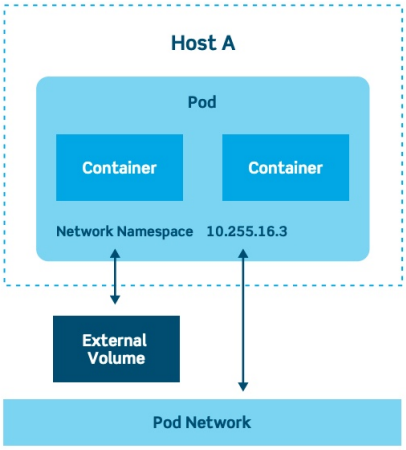

Figura 2.2: Architettura di un Pod in Kubernetes

2.6 Deployment

Un oggetto di tipo Deployment [3] permette di indicare tutte le istruzioni necessarie per rilasciare un set di pod all'interno di un cluster facendo convergere lo stato attuale delle risorse contenute al suo interno con lo stato desiderato. Quando questo oggetto è applicato nel cluster esso permette di definire un nuovo **ReplicaSet** [20].
Lo scopo di un ReplicaSet è mantenere un set stabile di pod di replica in esecuzione in qualsiasi momento. In quanto tale, viene spesso utilizzato per garantire la disponibilità di un numero specifico di Pod identici. Un oggetto di tipo ReplicaSet è composto da diversi campi, incluso un selettore che specifica come identificare

i Pod da replicare, un numero di repliche che indica quanti Pod devono essere mantenuti operativi e un modello di Pod che specifica i dati delle nuove repliche che deve creare per soddisfare il numero dei criteri di replica.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: custom-namespace
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Listing 2.2: Easempio di un oggetto Deployment

Lo scopo di un ReplicaSet [20] è mantenere un set stabile di pod di replica in esecuzione in qualsiasi momento. In quanto tale, viene spesso utilizzato per garantire la disponibilità di un numero specifico di Pod identici.

2.7 StatefulSet

Un oggetto StatefulSet [24] è usato per gestire le applicazioni che devono conservare un certo stato. Gestisce la distribuzione e il ridimensionamento di un set di pod e fornisce garanzie sull'ordine e l'unicità di questi pod.
Analogamente per quanto vale per i Deployment, uno StatefulSet gestisce multipli pod basati sullo stesso container. Ma a differenza di un Deployment, uno StatefulSet mantiene un'identità permanente per ciascuno dei relativi pod. Questi pod sono creati con le stesse specifiche, ma non sono intercambiabili in quanto ognuno ha un identificatore persistente che mantiene durante qualsiasi riprogrammazione.

2.8 Operatore

Un operatore [2] rappresenta un'estensione software di Kubernetes che utilizza risorse personalizzate per gestire le applicazioni e i loro componenti.
Gli operatori sono quindi quei componenti che si occupano di controllare periodicamente se ci sono differenze tra lo stato attuale delle risorse del cluster e lo stato desiderato, e nel caso provvedere a ripristinare lo stato desiderato, ad esempio se una risorsa non è più disponibile l'operatore provvede a ripristinarla o crearne una analoga.
Sotto alcuni aspetti un operatore in Kubernetes può essere associato ad un operatore umano che gestisce un insieme di servizi, in quanto entrambi rappresentano entità che hanno una profonda conoscenza di come dovrebbe comportarsi il sistema, come implementarlo e come reagire in caso di problemi. L'unità di codice che esegue il controllo è chiamata **Controller**, mentre il processo di ripristino dello stato desiderato prende il nome di **Riconciliazione**.

2.8.1 Scrittura di un operatore personalizzato

Se Kubernetes non espone un operatore specifico per implementare un comportamento desiderato c'è la possibilità di creare un proprio operatore personalizzato, l'implementazione è possibile utilizzando un qualsiasi linguaggio di programmazione che possa fungere da client per l'API di Kubernetes, ad esempio il Go.
In questo elaborato saranno presentati alcuni esempi di operatori personalizzati, che sono stati oggetto di lavoro durante l'attività di tesi, questi operatori sono stati implementati utilizzando le librerie del framework Kubebuilder.

2.9 Docker

Docker [4] rappresenta uno dei più noti sistemi open source per la gestione dei container in ambiente **Linux** e non solo.
La containerizzazione è un approccio moderno nello sviluppo e nel rilascio del software basato sull'architettura granulare dei microservizi e su un metodo come il Development and Operations (DevOps), ossia una metodologia basata su multiple unità di sviluppo del software, che permette agli sviluppatori di lavorare in contemporanea sulla singola componente di una struttura più ampia, anche adoperando tipologie di codice differente.
Docker si inserisce in questo contesto introducendo inoltre i vantaggi della **portabilità** e della **leggerezza**, la portabilità deriva dal suo processo di distribuzione basato su file di immagine contenenti tutto l'occorrente per creare un container, per quanto riguarda la leggerezza, questa dipende dalla capacità dei container di

sfruttare il kernel del sistema operativo offrendo la possibilità ad un calcolatore fisico di ospitare parecchi container.

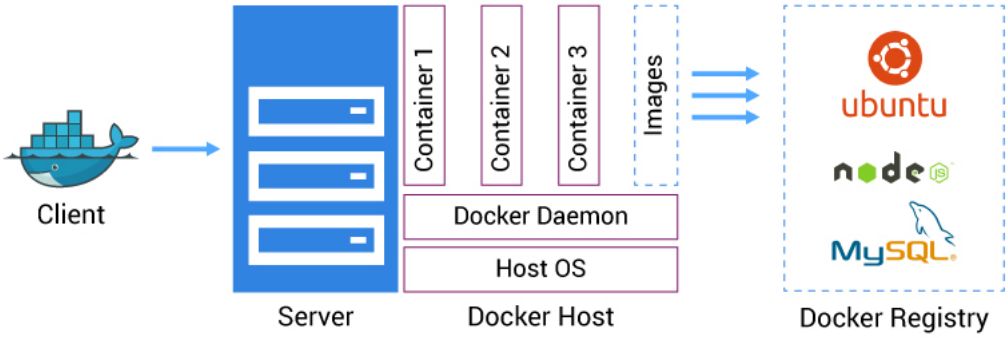

Figura 2.3: Architettura di Docker

2.9.1 Confronto con Kubernetes

Docker e Kubernetes sono spesso messi a confronto, ma in realtà sono strumenti complementari in quanto possono essere usati in fasi separate dello stesso sviluppo. Nello specifico Docker può fornire la distribuzione di container autosufficienti che costituiscono i componenti base dei pod, ossia le unità base di Kubernetes.

Capitolo 3

CrownLabs

3.1 Introduzione

Il progetto CrownLabs [13] è nato durante il primo lockdown del 2020 grazie al lavoro di un gruppo di volontari del Politecnico di Torino con l'obiettivo iniziale di fornire terminali desktop remoti agli studenti utili per superare i limiti di inaccessibilità dei laboratori del Politecnico durante le fasi di chiusura. Fortunatamente la situazione pandemica è notevolmente migliorata rispetto al periodo in cui CrownLabs è nato e attualmente i laboratori del Politecnico sono tornati nuovamente accessibili. Tuttavia il progetto di CrownLabs è ancora vivo e non accenna a voler limitarsi al singolo contesto accademico; infatti il progetto si sta espandendo cercando di aumentare il proprio bacino d'utenza includendo sempre maggiori servizi cloud al suo interno. Con il sempre più progressivo utilizzo di tali servizi, e con il conseguente aumento del numero di studenti interessati al settore, CrownLabs mira a diventare una vera e propria palestra per utenti, dando loro l'opportunità di poter costruire, manutenere e utilizzare servizi cloud.
Di seguito si riportano alcuni utilizzi di CrownLabs che si vanno ad aggiungere all'iniziale funzionamento che era stato ideato per il progetto.

Esami con CrownLabs A partire da Settembre 2021 CrownLabs ha dato la possibilità di svolgere gli esami mettendo a disposizione a ciascuno degli studenti candidati una istanza virtuale dove poter produrre l'elaborato da consegnare. In questo periodo gli esami sono stati svolti da remoto per via delle restrizioni dovute alla pandemia, e CrownLabs ha voluto offrire un'alternativa ai software indicati dal Politecnico per lo svolgimento degli esami.
L'appello di Informatica del primo anno del corso di laurea triennale di Ingegneria è stato il primo esame svolto con CrownLabs. La traccia d'esame richiede di produrre un programma scritto in linguaggio Python, perciò CrownLabs ha fornito ad ogni

studente un container con il software PyCharm in esecuzione, software utilizzato per la scrittura del codice Python.

Ulteriori servizi offerti Nella sua idea di espansione CrownLabs intende offrire agli utenti la possibilità di eseguire applicazioni esterne che esulano dal contesto della didattica del Politecnico. Alcuni esempi di applicazioni sono Visual Studio Code, Jupyter Notebooks e KubeFlow, e si vuole dare la possibilità agli utenti di poterle eseguire nelle stesse modalità di come viene fatto su un tradizionale computer Desktop.

Le prossime sezioni forniscono una breve panoramica dell'infrastruttura di CrownLabs e di alcuni concetti basilari su cui si è incentrata l'attività di tesi.

3.2 Infrastruttura di CrownLabs

L'infrastruttura generale di CrownLabs si basa su due componenti principali ad alto livello:

- **Dashboard**, che costituisce il frontend dell'applicazione esponendo le risorse personalizzate e i servizi offerti da CrownLabs attraverso un'interfaccia grafica. Di recente l'interfaccia di CrownLabs ha subito forti cambiamenti, che la hanno resa molto più intuitiva per l'utente e adattabile ad ogni dispositivo e risoluzione, compresi i dispositivi mobile. L'interfaccia consente agli utenti della piattaforma di esplorare le workspace in cui sono registrati, generare nuovi terminali e connettersi ad essi. Inoltre, gli utenti con maggiori permessi, rappresentati in questo contesto dai docenti del Politecnico, hanno la possibilità di creare, aggiornare ed eliminare sia i template dei terminali che gli utenti stessi, gestendo efficacemente gli ambienti disponibili e le autorizzazioni concesse per accedere al sistema. Il frontend è costituito da un'applicazione React, che utilizza i componenti grafici di Ant Design, ed è sviluppata in linguaggio Typescript.
- **Logica del backend**, che fornisce le diverse funzionalità lato server di CrownLabs ed è costruita attraverso l'implementazione di operatori Kubernetes personalizzati, le prossime sezioni illustrano maggiori dettagli di questi operatori.

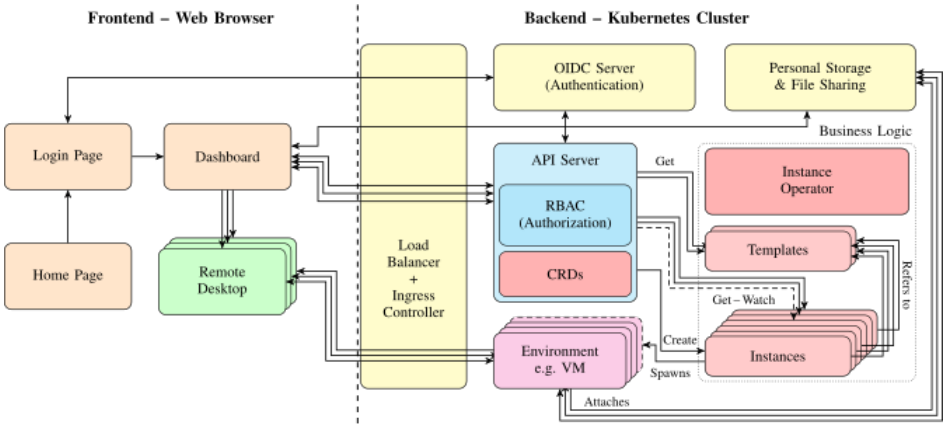

Figura 3.1: Architettura generale di CrownLabs

3.3 Tenant e Workspace

Un Tenant e una Workspace rappresentano rispettivamente un utente CrownLabs (studente o docente) e un'area di lavoro che può accomunare più utenti, ad esempio un corso frequentato da più studenti, di quì in avanti ogni volta che si nomina il **Tenant** si farà riferimento a un utente CrownLabs.
Un Tenant può essere iscritto a una o più Workspace, e ogni Workspace gli darà la possibilità di accedere a diversi template di istanze che egli può creare e utilizzare. I docenti hanno inoltre la possibilità di visualizzare ed accedere a tutte le istanze degli studenti iscritti al suo corso (workspace) di competenza.
A livello implementativo Tenant e Workspace sono due entità definite tramite delle CRD e l'operatore che ha il compito di ripristinare periodicamente lo stato desiderato per queste due risorse è il Tenant Operator.

3.4 Istanze

Le istanze costituiscono i terminali remoti che l'utente può creare e utilizzare, l'unico requisito necessario per sfruttare questo servizio è disporre di un browser e una connessione internet, dopodichè spetta a CrownLabs fornire le risorse comptazionali necessarie, e garantire il corretto funzionamento degli ambienti remoti.
I terminali possono anche essere provvisti di interfaccia grafica (non tutti) in modo da poter essere considerati come veri e propri ambienti Desktop, in tal caso l'utente potrà controllare l'interfaccia grafica da remoto tramite noVNC.
Esistono due tipologie di istanze che possono essere create:

- **Container**, la cui descrizione coincide con quella fatta nel capitolo precedente

- **Virtual Machine (VM)**, questa tipologia di istanza è inusuale per un ambiente kubernetes-based, ma è stata introdotta in quanto soluzione funzionante nell'immediato, date le circostanze e le tempistiche in cui CrownLabs è stato sviluppato.
  Il loro utilizzo è stato reso possibile grazie all'integrazione di KubeVirt, un software open source che consente di eseguire una VM in ua piattaforma container. KubeVirt introduce una nuova CRD, ossia la Virtual Machine Instance (VMI) che permette di definire questa particolare tipologia di istanza. Le VM a loro volta si classificano in due tipi:

  – **Persistenti**, sono dotate di storage, questo permette di poter metterle in pausa una volta create, in modo da non consumare risorse computazionali del cluster, ma rimanere disponibili per un nuovo utilizzo ripartendo dall'esatta configurazione in cui ci si è fermati prima di metterle in pausa.

  – **Non persistenti**, non sono dotate di storage, perciò alla fine del loro utilizzo possono solo essere distrutte, non è possibile quindi salvare il loro stato.

Ogni istanza è caratterizzata da un **Template** che definisce il gruppo di appartenenza di ciascun modello di istanza offerto e il tipo di istanza, scegliendo tra Container e Virtual Machine persistenti o non, ogni Template a sua volta contiene un **Environment**, ossia la descrizione di tutte le specifiche tecniche dell'ambiente, quali l'immagine di sistema, le risorse computazionali riservate e le politiche di rete. Le risorse descritte in questa sezione sono gestite dall'Instance Operator.

3.5 Operatori Kubernetes utilizzati da CrownLabs

La logica di business backend è implementata tramite la realizzazione di operatori Kubernetes personalizzati, mentre il modello dati è definito tramite CRD.
Gli operatori principali di CrownLabs, su cui è incentrata parte di questa attività di tesi sono i seguenti:

- **Tenant Operator**, automatizza la gestione degli utenti in CrownLabs, includendo le fasi di registrazione dell'account, iscrizione ad una o più workspace e attribuzione di privilegi specifici per l'utilizzo della piattaforma. Le azioni eseguite dall'operatore comprendono la creazione di una risorsa Tenant associata ad un nuovo utente con i suoi dati anagrafici; la creazione o modifica di tutti gli oggetti del cluster legati al tenand, ad esempio namespace e Resource Quota (maggiori dettagli su questo oggetto saranno forniti nel prossimo capitolo); controllare se il Tenant risulta iscritto ad una Workspace

non più esistente e in tal caso rimuovere l'iscrizione; gestire i ruoli dell'utente all'interno della piattaforma; e infine gestire l'eliminazione di un utente da CrownLabs cancellando tutti gli oggetti a lui legati dal cluster. Il Tenant Operator comprende inoltre azioni per gestire gli oggetti del cluster legati alle Workspace. Il Controller di questo operatore viene innescato dopo ogni aggiornamento alle risorse a cui è agganciato, ossia Tenant e Workspace, e dopo ogni modifica agli oggetti gestiti dall'operatore stesso.

- **Instance Operator**, implementa la logica backend per gestire tutte le fasi di creazione generazione di nuovi ambienti di lavoro a partire dai template predefiniti. In questo operatore all'atto della creazione di un'istanza viene fatta la selezione circa la sua persistenza attraverso le informazioni contenute nel suo template. Nel caso di istanza persistente l'operatore istanzia un oggetto KubeVirt VM piuttosto che un oggetto VMI, questa scelta consente di delegare a KubeVirt stesso il processo di avvio e arresto di una VM. L'Instance Operator gestisce inoltre tutte le fasi di cancellazione di un'istanza, che l'utente può innescare attraverso la sua dashboard, eliminando dal cluster tutti gli oggetti inseriti in fase di creazione.

3.6 Altri componenti

Esistono ulteriori componenti esterni su cui CrownLabs si basa per gestire altri aspetti legati al suo funzionamento, quali gestione delle identità degli utenti, condivisione di file e monitoraggio delle prestazioni.
La gestione dell'autenticazione e autorizzazione degli utenti avviene attraverso l'identity provider di Keycloak, dotato di funzionalità avanzate e alquanto immediato da configurare, maggiori dettagli sono forniti nel prossimo capitolo.
La condivisione di file è utile per fornire un servizio di storage a cui gli utenti possono accedere sia dall'interno che dall'esterno dei loro terminali Desktop remoti, questo aspetto è gestito dall'integrazione di Nextcloud.
Infine il monitoraggio delle prestazioni del cluster avviene tramite le metriche rese disponibili da Prometheus, e la cui visualizzazione avviene attraverso la creazione di dashboard su Grafana.

Capitolo 4

Altri Strumenti open-source utilizzati

In questo capitolo sono illustrati gli altri software open-source protagonisti dell'attività svolta.

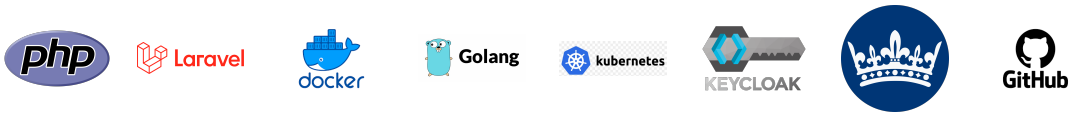

Figura 4.1: Principali strumenti open-source utilizzati

4.1 GitHub

GitHub [10] è il servizio più conosciuto di hosting per progetti software, e permette agli sviluppatori di archiviare il loro codice e tracciare e controllare le modifiche, questo servizio si basa sul sistema di **Git** [7].
Git è uno strumento gratuito di controllo di versione, esso permette quindi di avere traccia dello storico delle modifiche del codice sorgente di un qualsiasi progetto nel corso del tempo, una modifica viene identificata con il termine **commit**. Git è utilizzabile tramite Command-Line Interface (CLI), tuttavia GitHub si rivela particolarmente utile in quanto, oltre ad offrire un servizio di hosting di repository basato su cloud, espone un'interfaccia grafica che rende l'utilizzo di Git più intuitivo.
GitHub permette agli sviluppatori di rendere il loro codice sorgente disponibile e scaricabile dagli utenti all'interno di un **repository**, gli utenti possono poi proporre nuove modifiche al codice sorgente. La norma prevede che le modifiche siano

caricate in un nuovo **branch** nel repository, tramite i branch si possono creare molteplici flussi di sviluppo isolati tra di loro e focalizzati su obiettivi diversi.
Il codice sorgente di CrownLabs è consultabile all'interno dell'omonimo repository [21] dell'organizzazione **Netgroup Polito**, nelle successive sezioni sarà inoltre menzionato il repository **faveo-helpdesk** [22] che contiene il codice sorgente del software di Ticketing integrato in CrownLabs durante l'attività di tesi.
Le interazioni con questi repository sono avvenute sfruttando il sistema delle Pull Request (PR) [18], esse permettono ai proprietari del repository di discutere le modifiche proposte con gli altri collaboratori, ed eventualmente suggerire ulteriori cambi prima di procedere con l'applicazione delle modifiche al branch base, meglio conosciuto come **Master**.

4.2 Laravel

In questa sezione si introduce il framework **Laravel** [1], mediante il quale è stato sviluppato il software di ticketing integrato in CrownLabs, che sarà introdotto nel prossimo capitolo.
Laravel è un framework open-source PHP per applicazioni web, rilasciato nel 2011, esso si basa sul pattern Model-View-Controller (MVC) e presenta una sintassi espressiva ed elegante.
Laravel semplifica alcune delle attività più comuni dei progetti web, come autenticazione, autorizzazione, routing, sessioni, caching. Possiede inoltre un motore di templating integrato per le interfacce grafiche, chiamato Blade, un sistema di migrazione database e una propria interfaccia a riga di comando, chiamata Artisan CLI.

4.3 Keycloak

Keycloak [15] è un software open-source di Identity and Access Management (IAM), esso mira a verificare l'identità di un utente o di un sistema che richiede l'accesso ad un ambiente e valuta tutte le regole che indicano a quali funzionalità e risorse l'utente o sistema ha accesso.
Keycloak supporta il protocollo OpenID Connect e permette quindi di abilitare il SSO all'interno di applicazioni e servizi moderni. In questo software sono inoltre combinati sia un provider di identità che un server di autenticazione.
Nel contesto di CrownLabs ci si serve di questo prodotto per gestire la registrazione e l'autenticazione degli utenti all'interno della piattaforma, questo ha come vantaggio principale la centralizzazione dell'autenticazione per tutti i servizi offerti e non preoccuparsi degli aspetti di sicurezza in fase di sviluppo, che sono gestiti da Keycloak stesso.
Le sezioni seguenti illustrano i concetti di **Realm** e **Client** in Keycloak, che saranno ripresi all'interno del prossimo capitolo incentrato sul Ticketing.

4.3.1 Realm

Un **Realm** è uno spazio in cui si gestiscono tutti gli oggetti, inclusi utenti, applicazioni, ruoli e gruppi, un utente appartiene quindi ad un realm, e vi può fare accesso.
In quest'ottica CrownLabs può essere identificato come il realm all'interno del quale gli utenti si autenticano. La seguente figura mostra l'interfaccia che permette di configurare le impostazioni di un realm.

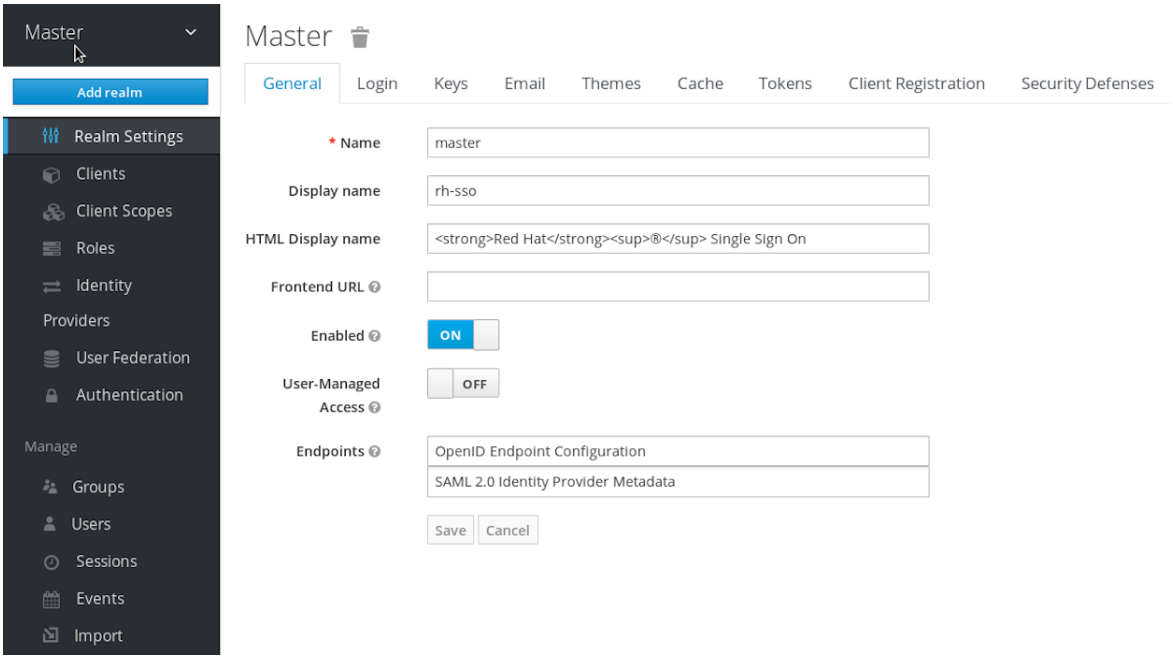

Figura 4.2: Impostazioni di configurazione per un Realm

4.3.2 Client

I client sono entità che possono contattare il server di Keycloak per autenticare un utente. Nella maggior parte dei casi, i client sono applicazioni e servizi che desiderano utilizzare Keycloak per proteggersi e fornire una soluzione SSO.
I client possono anche essere entità che desiderano semplicemente richiedere informazioni sull'identità o un token di accesso in modo da poter invocare in modo sicuro altri servizi sulla rete protetti da Keycloak. La seguente figura mostra l'interfaccia grafica che permette di configurare un nuovo client.

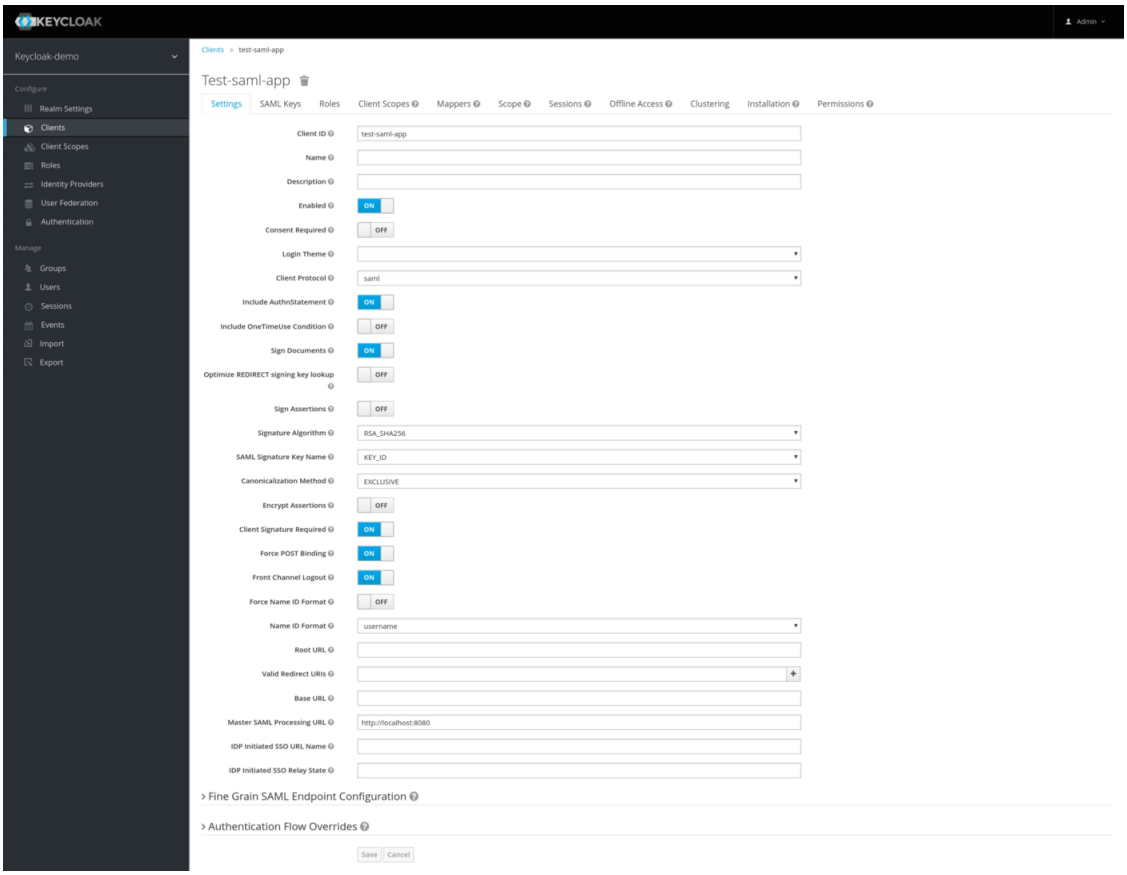

Figura 4.3: Impostazioni di configurazione per un Client

Capitolo 5

Faveo Helpdesk

Questo capitolo presenta l'integrazione in CrownLabs di **Faveo Helpdesk** [5], un software open source sviluppato da **Ladybird Web Solution** mediante l'utilizzo del framework Laravel basato sul linguaggio PHP presentato nella sezione 4.2. Lo scopo è offrire un sistema di helpdesk automatizzato per la gestione dell'assistenza clienti mediante l'utilizzo di ticket. I principali vantaggi offerti da un sistema di ticketing sono di seguito elencati:

- **Miglior distribuzione delle comunicazioni**, ogni membro del team di supporto riceverà le comunicazioni legate alla sua area di competenza.
- **Archiviazione delle segnalazioni in un sistema centralizzato**, le richieste degli utenti sono raccolte in un unico posto, e suddivise per categoria, in modo da gestirle in modo più efficiente.
- **Interazioni consolidate in un thread**, l'assistenza a un utente può coinvolgere diversi agent, ed essere erogata in più step, le comunicazioni tra agent e utente sono tuttavia memorizzate nello stesso ticket, quindi risultano accessibili a tutti.

Alcuni dei principali moduli tipici di un software di helpdesk sono indicati nella segentue figura. Tuttavia solo alcuni di questi moduli saranno approfonditi e utilizzati all'interno dell'elaborato.

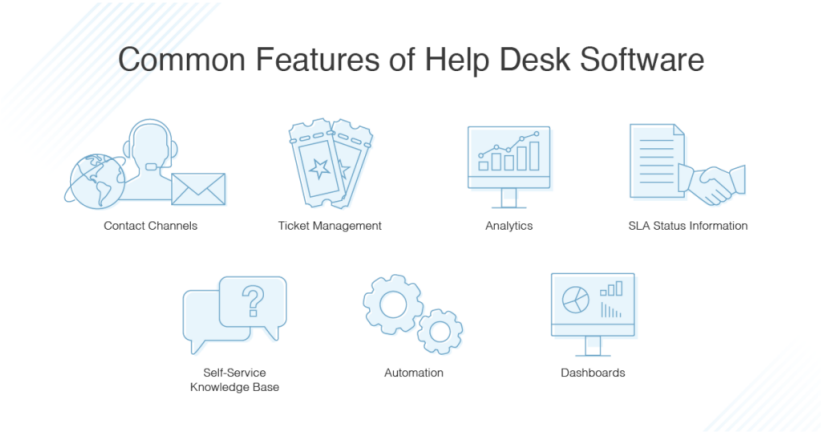

Figura 5.1: Caratteristiche di un software di ticketing

5.1 Caratteristiche di Faveo Helpdesk

Faveo Helpdesk è una soluzione di Helpdesk che si rivolge alle startup e alle piccole e medie imprese, che permette di fornire assistenza ai clienti tramite l'apertura di ticket. Il sistema assegna un id univoco ad ogni ticket, si possono assegnare diversi livelli di priorità ai ticket e impostare risposte automatiche basate su modelli a determinati tipi di richieste. La soluzione include anche un'interfaccia grafica dove i clienti possono accedere e controllare lo stato del loro ticket.
Faveo consente di assegnare ticket a dipendenti o dipartimenti specifici, in quanto è possibile assegnare più utenti ai ticket. È inoltre possibile creare note interne per comunicare con gli altri membri dello staff sullo stato del ticket.

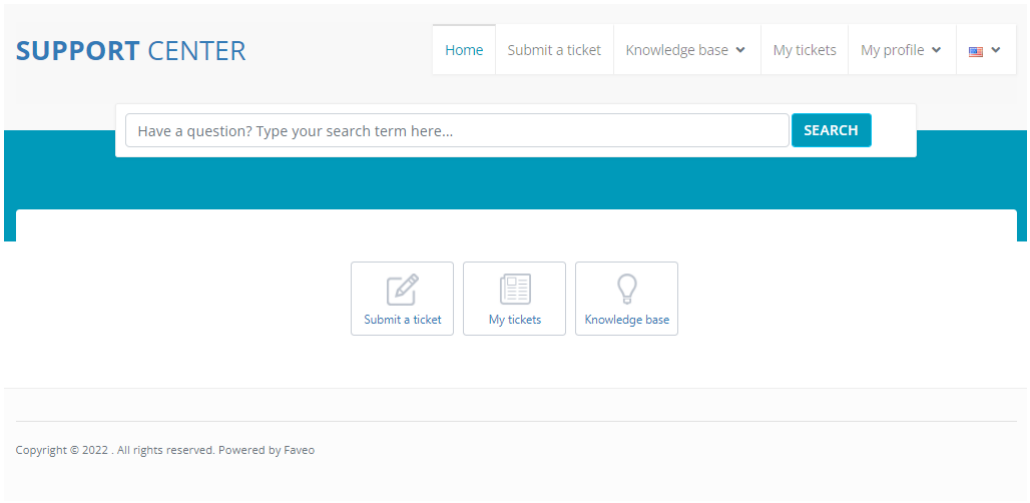

Figura 5.2: Home Page di Faveo Helpdesk

5.1.1 Ruoli degli utenti

Gli utenti in Faveo sono divisi in tre ruoli, ogni ruolo garantisce privilegi progressivamente maggiori rispetto al precedente, essi sono:

- **User**, rappresenta il ruolo base, e di conseguenza la cerchia più estesa degli utenti, ossia coloro che necessitano di assistenza.
  Loro possono creare ticket e/o consultare i ticket già aperti e hanno accesso solo alle sezioni all'interno della home page per poter effettuare queste operazioni

- **Agent**, rappresenta chi deve fornire assistenza agli utenti, perciò sono coloro che rispondono ai ticket, l'applicazione permette anche ad un agent di aprire un ticket.
  Per gli agent, oltre alle sezioni della home page, è previsto un ulteriore pannello, all'interno del quale possono visualizzare e gestire i ticket aperti, e visualizzare le informazioni relative agli account di ciascun utente

- **Admin**, questa è la tipologia di utente con i privilegi più alti, di base può effettuare il lavoro di un comune agent, ma in più ha accesso al pannello di amministrazione, dove può modificare le configurazioni di sistema e gestire l'assegnazione dei ticket a un gruppo di agent.
  Maggiori dettagli sulle impostazioni di sistema in Faveo sono forniti nella prossima sezione

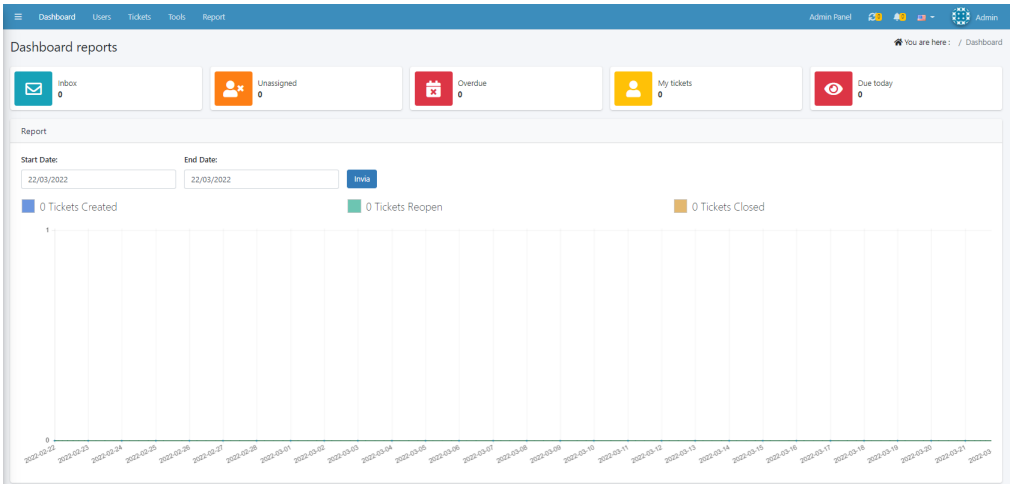

Figura 5.3: Pannello Agent di Faveo Helpdesk

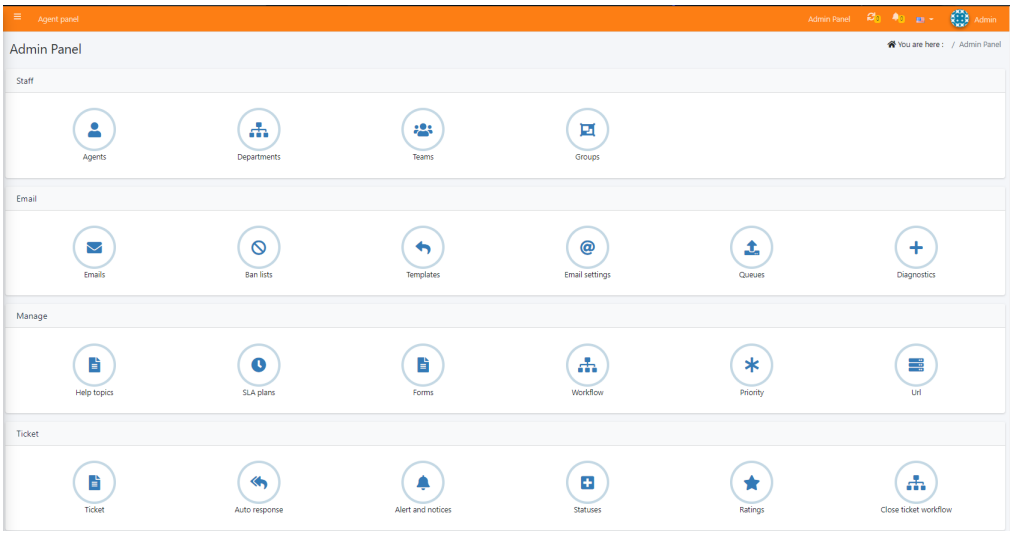

Figura 5.4: Pannello Admin di Faveo Helpdesk

5.1.2 Impostazioni di sistema

Le configurazioni di sistema alle quali un utente di tipo amministratore ha accesso sono molteplici, e variano a seconda della versione di Faveo utilizzata.
In questo caso si utilizza la versione gratuita e open source, il cui codice sorgente è disponibile su un repository pubblico in GitHub, di proprietà di Ladybird Web Solution.
Tra i componenti di configurazione presenti in questa versione ci concentriamo sui seguenti.

Help Topic Gli help topic sono gli argomenti messi a disposizione dalla piattaforma sui quali un utente può chiedere assistenza.
Idealmente ad ogni topic è associato un gruppo di agent dotati delle conoscenze necessarie per poter fornire assistenza, un gruppo può essere costituito da uno o più membri.

Dipartimento Un dipartimento definisce una macro area di supporto per gli utenti, può essere associato a uno o più help topic ed è utile per lo staff di supporto per formare i gruppi di agent. Ogni agent (o admin che ne esegue il lavoro) può visualizzare e gestire solo i ticket relativi ad help topic associati al dipartimento di cui egli fa parte.
Questo è utile per garantire una miglior distribuzione delle comunicazioni e fare in modo che i membri dello staff interagiscano esclusivamente con i ticket di loro competenze.

Social Login Questo componente permette di dare la possibilità agli utenti di effettuare la procedura di login sulla piattaforma di Faveo in modalità alternative rispetto al classico inserimento delle credenziali **Username** e **Password**.
Faveo supporta il login attraverso gli identity provider di **Facebook**, **Google**, **GitHub**, **Twitter**, **Linkedin** e **Bitbucket**.

Mail Sender Il mail sender è quel servizio che permette di inviare notifiche tramite Email agli utenti della piattaforma quando si verificano alcune tipologie di eventi, ad esempio:

- **Creazione di un nuovo ticket**, in questo caso viene inviata una mail all'utente che ha creato il ticket, come conferma di avvenuta creazione, e una mail agli agent di competenza.
- **Registrazione di un utente**, si invia una mail all'utente appena registrato contenente la password per il primo login, questo comportamento è stato

modificato per l'integrazione in CrownLabs, le sezioni successive illustrano maggiori dettagli

- **Risposta a un ticket**
- **Chiusura di un ticket**
- **Assegnazione di un ticket a un agent**

Per ognuno di questi eventi è possibile personalizzare il template della notifica mail che viene inviata.
Il servizio di mail sender è configurabile inserendo le informazioni relative ai server che si vogliono usare per la ricezione e l'invio di mail.

5.1.3 Configurazione per il primo utilizzo

Per poter rendere disponibile il servizio di Faveo sul proprio sistema operativo bisogna seguire le istruzioni indicate nell'apposita sezione della documentazione presente sul repository GitHub.
Una volta completata l'installazione il sistema prevede alcuni step di configurazione da effettuare prima di procedere con il primo utilizzo, nell'ordine questi step eseguono le seguenti attività:

- Controllo che il sistema soddisfa i requisiti per poter eseguire Faveo
- Impostazione della lingua e dell'area geografica.
- Inserimento del database, questo è lo step più lungo, si richiede di fornire le generalità del database in cui memorizzare i dati dell'applicazione.
  Il database deve essere di tipo MySql con versione 5.0 o superiore, e deve essere vuoto, l'utente deve inserire le stringhe relative al nome del database, username e password dell'utente proprietario del database, e infine hostname e porta del server che ospita il database.
  Una volta inserite queste stringhe il sistema effettua un controllo sulla validità dei dati e sull'assenza di entità nel database. Se il controllo ha esito positivo il sistema provvede ad effettuare la migrazione, ossia la procedura di creazione dello scheletro delle entità nel database a partire dal codice sorgente in Laravel PHP.
- Creazione del primo utente Admin, in quest'ultimo step si effettua la registrazione del primo utente della piattaforma, si inseriscono quindi i dati anagrafici personali e le credenziali di accesso. Dopodichè si effettua il login e da quel momento l'utente Admin tramite il pannello a lui dedicato può effettuare tutte le configurazioni di sistema descritte in precedenza.

5.2 Utilizzo in CrownLabs

Con il passare del tempo CrownLabs sta aumentando il suo bacino di utenza e il numero di servizi offerti, questo porta ad un aumento delle casistiche in cui un utente può avere bisogno di richiedere supporto e ad una maggior necessità da parte di chi fornisce il servizio di ricevere feedback.
Tutto questo dimostra l'utilità di introdurre un sistema che permetta di gestire le comunicazioni tre utente e staff tecnico e che sia accessibile direttamente dall'interno della piattaforma di CrownLabs, seguendo il modello di quanto è stato realizzato all'interno del Portale della Didattica del Politecnico di Torino.
L'utente può accedere alla piattaforma di Faveo Helpdesk tramite un pulsante sulla barra di navigazione della sua dashboard CrownLabs.

Figura 5.5: Pulsante Support per accedere a Faveo da dentro CrownLabs

A questo pulsante è associato l'URL per l'applicazione web di Faveo, al suo interno l'utente può consultare gli eventuali ticket aperti in precedenza, oppure creare un nuovo ticket selezionando uno tra i seguenti argomenti:

- **Issues report**, per segnalare eventuali problemi con un servizio di CrownLabs (questo topic per il momento è generico, però all'occorrenza può essere diviso in più sotto-argomenti)
- **Graphical interface**, La dashboard di CrownLabs ha subito un grande cambiamento negli ultimi mesi, in questa sezione l'utente può ad esempio richiedere chiarimenti sull'utilizzo dell'interfaccia grafica, segnalare eventuali malfunzionamenti o proporre nuovi miglioramenti
- **Other**, Selezionando questo argomento l'utente può aprire un ticket riguardante un qualsiasi argomento che non rientra nelle precedenti casistiche

Per notificare gli utenti è stato configurato il servizio di mail sender con le informazioni relative ai server IMAP e SMTP utilizzati da CrownLabs rispettivamente per la ricezione e l'invio delle mail.

5.3 Single Sign-On

Partendo dal codice sorgente disponibile sul repository GitHub di Ladybird Web Solution [23] la prima attività svolta è stata integrare in Faveo un sistema per

permettere a un utente CrownLabs di non dover effettuare una registrazione o un login su una piattaforma esterna. Quando un utente già loggato su CrownLabs accede a Faveo tramite l'apposito collegamento nella dashboard si possono verificare due casi, e il corretto funzionamento prevede che

- Se l'utente non ha un account su Faveo il sistema ne crea uno con gli stessi dati anagrafici relativi al suo account in CrownLabs, e usa il suo indirizzo mail per impostare lo username
- Se l'utente ha già un account su Faveo il sistema si occupa di effettuare la procedura di login

In entrambi i casi l'utente è rediretto sulla home page di Faveo, e riesce ad accedervi senza conoscere i dettagli dell'account che è stato creato dal sistema, questa proprietà è meglio conosciuta come SSO [17].
Per rendere il processo di registrazione del tutto trasparente all'utente è stato rimosso il template di notifica utilizzato per inviare la conferma di registrazione e la password temporanea, in quanto a questo punto diventano informazioni superflue per l'utente. Il sistema di SSO per Faveo è stato realizzato con il supporto di **Keycloak**, i cui dettagli sono stati illustrati nella sezione 4.3.
La prima attività svolta per aggiungere Faveo nella lista dei servizi offerti da CrownLabs in cui l'autenticazione è centralizzata in un unico Identity Provider consiste quindi nella creazione del client **ticketing** all'interno del realm di CrownLabs.

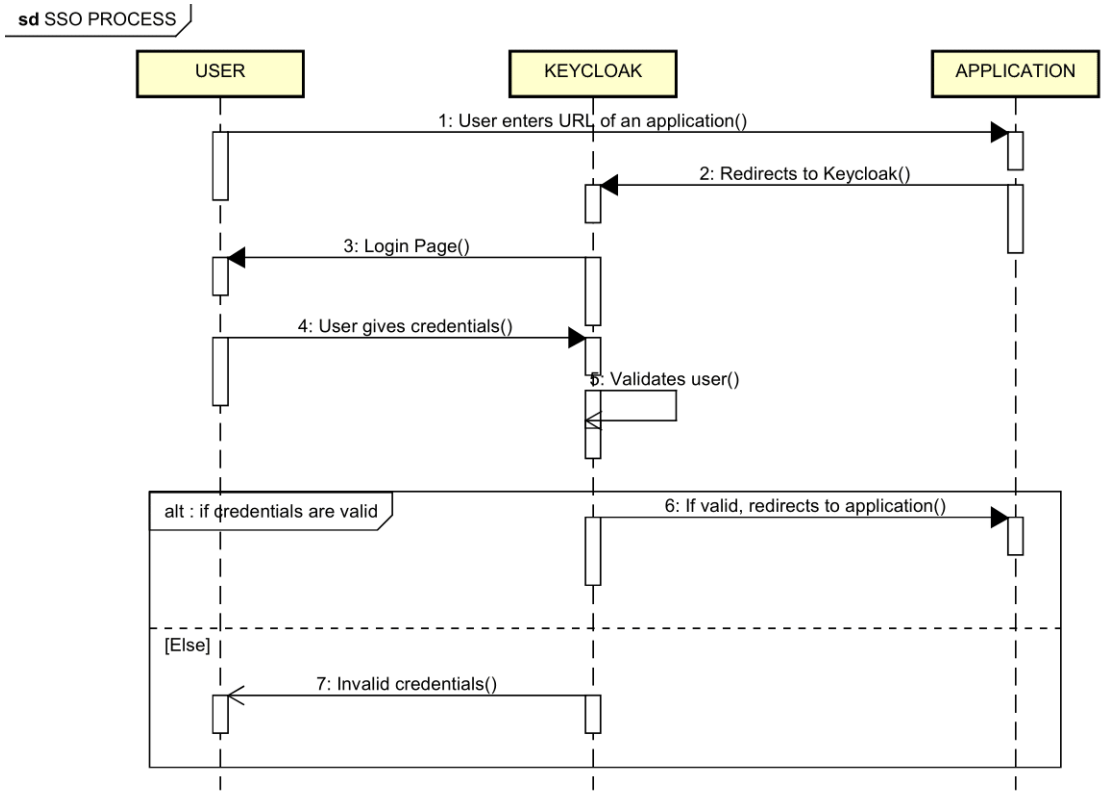

Figura 5.6: Sequence Diagram per il processo di SSO

5.3.1 Reindirizzamento del Login

Il passaggio successivo comprende l'implementazione della logica che permette il reindirizzamento del browser di un utente al server di autenticazione Keycloak dove egli può inserire le proprie credenziali valide per CrownLabs.
Questo comportamento è stato ottenuto sfruttando il Social Login di Faveo introdotto nella precedente sezione. Si può configurare un nuovo Identity Provider all'interno del pannello Admin di Faveo inserendo nella sezione di Social Login i seguenti parametri:

- **Client ID** = Id del client Keycloak, nel nostro caso la stringa "ticketing"
- **Client Secret** = Secret del client di Keycloak, impostato in fase di creazione del client stesso
- **URL di reindirizzamento** = URL del server di autenticazione di Keycloak

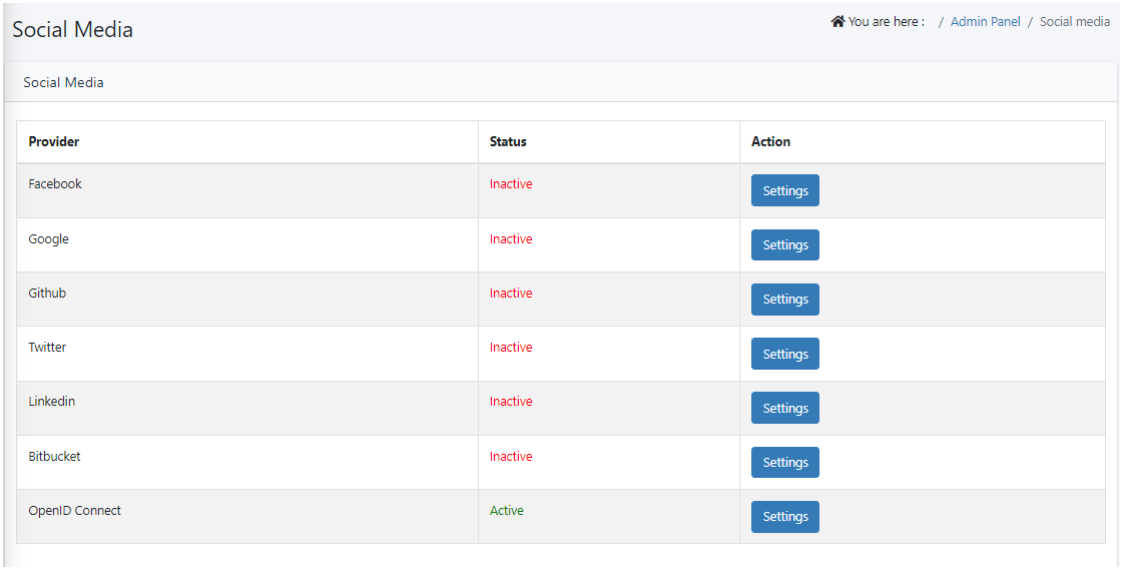

Figura 5.7: Sezione Social Login di Faveo con l'aggiunta del provider di OpenID Connect

Il reindirizzamento avviene nel momento in cui l'utente pigia il pulsante per accedere a Faveo dalla dashboard di CrownLabs, essendosi già loggato in precedenza su CrownLabs (ha dovuto effettuare il login per accedere alla dashboard) si trova di fatto già loggato in Faveo senza necessità di re-inserire le sue credenziali.
Nel caso in cui un utente CrownLabs non ha ancora un account associato su Faveo, la creazione di quest'ultimo avviene sempre ad opera di Keycloak, in ogni

caso l'utente è ignaro di questi processi, e può da subito usufruire del servizio di Helpdesk. Il seguente box fornisce uno pseudocodice che illustra la logica di SSO implementata nel codice sorgente.

```
user visits faveo base url;
if(socialProviders.checkActive(openidConnect))
    redirect to keycloak auth url;
    if(user is not logged in crownLabs)
        insert crownLabs auth credentials;
    store user info from provider callback into local variables;
    if(faveoUsers.isNotPresent(user))
        user = new User(firstName, lastName, email, userName, role);
        faveoUsers.add(user);
    authController.login(user);
else
    ask user for a traditional login or signup;
```

Il codice PHP originale può essere trovato sul repository GitHub di Netgroup Polito riservato a Faveo [22].

5.4 Deploy sul cluster

Avendo completato l'integrazione del sistema di SSO questa sezione presenta la modalità con cui Faveo Helpdesk è stato reso disponibile per l'utilizzo nel contesto di CrownLabs.

5.4.1 Dockerizing Faveo

Per rendere l'applicazione di Faveo eseguibile in un ambiente Kubernetes è stata costruita un'immagine Docker che contiene il codice sorgente PHP in aggiunta delle definizioni per librerie e dipendenze necessarie.
La costruzione dell'immagine Docker avviene attraverso un **Dockerfile** [9], ossia un file di testo che contiene una sequenza di comandi, che nel nostro caso eseguono queste azioni:

- Aggiungere le dipendenze, per poter eseguire Faveo sono necessari i seguenti componenti:
  - **PHP** versione 7.1 o superiore, con le estensioni Imap, Mbstring, Mcrypt, OpenSSL, PDO, Tokenizer, XML, Zip
  - **MySQL** versione 5.0 o superiore, non è inserito nell'immagine Docker, la gestione del database sarà illustrata nella prossima sezione
  - **Web Server**, nel nostro caso è stato usato **Nginx**
  - **Composer** versione 1.7
- Inserire il codice sorgente
- Configurare Nginx per hostare Faveo
- Compilare l'applicazione di Faveo dal codice sorgente usando Composer

L'immagine Docker risultante da queste operazioni è disponibile sul repository DockerHub di CrownLabs.

5.4.2 Configurazione per Kubernetes

In questa sezione si illustrano i file yaml di configurazione usati per deployare Faveo, ossia costruire ed esporre sul cluster e sul web una unità esecutiva per l'applicazione.
Le risorse configurate in questi file saranno poi applicate all'interno del namespace **crownlabs-ticketing**.

Database Come specificato in precedenza tra i requisiti necessari per eseguire Faveo vi è la presenza di un database MySQL inizialmente vuoto. Per soddisfare questo requisito è stato creato un file manifest in YAML che sfrutta le caratteristiche di un operatore MySQL sviluppato da **Bitpoke** [16]. Applicando questo file manifest sul cluster di CrownLabs all'interno del namespace creato in precedenza si può creare un container che all'interno contiene un cluster MySQL riservato per lo scopo di Faveo.
L'operatore MySQL di Bitpoke ha permesso di costruire questo cluster in modalità **High Availability** [19], ossia un'infrastruttura con più repliche ridondanti gestite attraverso uno StatefulSet. Questa configurazione torna utile nel momento in cui Kubernetes rileva un malfunzionamento in una delle repliche, in quel caso si sfruttano le repliche secondarie per mantenere il servizio disponibile ed evitare la perdita di dati.
In parallelo al container con il cluster MySQL è stata deployata una risorsa di tipo **Secret** contenente le credenziali dell'utente proprietario del database e il nome del database stesso, questa risorsa è necessaria per poter creare il cluster MySQL.

Service e Ingress Queste due risorse servono per esporre il container di Faveo sul web, assegnando un indirizzo IP univoco all'interno del cluster e una URL per raggiungerlo dall'esterno, questa URL è stata poi legata al pulsante sulla dashboard in CrownLabs che permette agli utenti di accedere all'applicazione di Faveo.

Configmap Questa risorsa contiene le variabili d'ambiente per Faveo, tra tutte le variabili ci sono alcune che dipendono dal contesto in cui si vuole deployare Faveo, e sono le seguenti:

- **DB-USERNAME**, **DB-PASSWORD** e **DB-DATABASE**, ossia credenziali del proprietario del database e nome del database, sono rintracciabili dal Secret legato al cluster MySQL
- **ADMIN-USERNAME**, username dell'utente amministratore per Faveo
- **ADMIN-PASSWORD**, password dell'utente amministratore per Faveo

Le variabili di ambiente quì definite vengono iniettate direttamente all'interno del container, questo permette di oltrepassare gli step di configurazione per il primo utilizzo di Faveo, in modo da evitare il caso in cui, per un eventuale malfunzionamento del container, si debbano rieseguire tutti gli step.

Deployment Questa risorsa contiene le configurazioni necessarie per costruire e rilasciare il container su cui va in esecuzione l'applicazione di Faveo. Tra queste configurazioni vi sono la quantità di risorse computazionali (CPU e RAM) richieste da Faveo e l'immagine Docker da cui si costruisce l'unità esecutiva dell'applicazione.
In aggiunta al container principale vi è un **Init Container**, ossia un container specifico eseguito prima del container principale, ed è usato per migrare sul database MySQL le entità definite nel codice sorgente di Faveo sfruttando i comandi della Artisan CLI di Laravel.
Si riporta a titolo di esempio il manifest in YAML creato per configurare il Deployment di Faveo e creare quindi il Container e l'Init Container appena descritti:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/name: faveo
    app.kubernetes.io/part-of: faveo
  name: faveo
  namespace: crownlabs-ticketing
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: faveo
      app.kubernetes.io/part-of: faveo
  template:
    metadata:
      name: faveo
      labels:
        app.kubernetes.io/name: faveo
        app.kubernetes.io/part-of: faveo
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - topologyKey: "kubernetes.io/hostname"
            labelSelector:
              matchExpressions:
              - key: app.kubernetes.io/name
                operator: In
                values:
                - faveo
              - key: app.kubernetes.io/part-of
                operator: In
                values:
                - faveo
      volumes:
      - name: php-configmap-volume
        configMap:
          name: faveo-php-configmap
          items:
          - key: .env
            path: .env
      initContainers:
      - name: init-faveo
        image: crownlabs/faveo:v1.11.1-crown
        command: ["/bin/sh","-c"]
        args: ["php artisan migrate --force && php artisan db:seed
    --force || true"]
```

```
48          volumeMounts:
49          - name: php-configmap-volume
50            mountPath: /usr/share/nginx/.env
51            subPath: .env
52        containers:
53        - image: crownlabs/faveo:v1.11.1-crown
54          name: faveo
55          ports:
56          - containerPort: 80
57            name: http
58          resources:
59            requests:
60              memory: "200Mi"
61              cpu: "500m"
62            limits:
63              memory: "512Mi"
64              cpu: "1"
65          volumeMounts:
66          - name: php-configmap-volume
67            mountPath: /usr/share/nginx/.env
68            subPath: .env
```

Listing 5.1: Manifest per il Deployment di Faveo Helpdesk

Applicando questo file manifest, oltre agli appositi container, viene creato anche un oggetto ReplicaSet che contiene le informazioni sul numero di repliche del container principale di Faveo, impostato a 1, e su come queste repliche devono essere mantenute attive.
Tutti i file yaml di configurazione sono memorizzati nella directory **Infrastructure** del progetto CrownLabs in GitHub, in questa directory ci sono tutte le configurazioni dei componenti esterni di cui CrownLabs usufruisce.

Capitolo 6

Assegnazione risorse

Questo capitolo introduce il nuovo modello di contabilità delle risorse assegnate a ciascun utente di CrownLabs.
In Kubernetes quando più utenti o team condividono un cluster con un numero fisso di nodi, c'è il rischio che qualcuno possa utilizzare più della sua giusta porzione di risorse. Gli amministratori del cluster possono far fronte a questo problema servendosi di uno strumento chiamato **Resource Quota** [12], nella prossima sezione entriamo più nel dettaglio di questo strumento per fornire una breve panoramica.

6.1 Resource Quota

Un oggetto di tipo Resource Quota viene creato dall'amministratore del cluster, per fornire le regole per limitare il consumo di risorse e il numero di oggetti che possono essere creati all'interno di un namespace.
L'utente viene quindi monitorato dal sistema affinchè non vada ad eccedere i limiti che sono stati assegnati al suo (o ai suoi) namespace di appartenenza.
Le tipologie di vincoli che le Resource Quota possono imporre sono molteplici, in questa sezione ci soffermiamo sui vincoli usati all'interno del contesto di CrownLabs, che sono i seguenti:

- **requests.cpu**, la somma dei CPU requests per tutti i pod all'interno del namespace non può eccedere questo valore riservate per il pod
- **requests.memory**, la somma dei memory requests per tutti i pod all'interno del namespace non può eccedere questo valore, la parola chiave memory indica la memoria RAM
- **limits.cpu**, la somma dei CPU limits per tutti i pod all'interno del namespace non può eccedere questo valore

- **limits.memory**, la somma dei memory limits per tutti i pod all'interno del namespace non può eccedere questo valore
- **count/<resource>**, con questa sintassi si può impostare il numero totale di oggetti di tipo <resource> che si possono creare all'interno del namespace. <resource> può rappresentare anche una qualsiasi CRD, in CrownLabs questa tipologia di vincolo è usata per limitare il numero massimo di istanze che un utente può creare, perciò la sintassi adottata è **count/instances.crownlabs.polito.it**

In questa sintassi la parola chiave *requests* indica la quantità minima di risorse che deve essere garantita per i pod all'interno del namespace, mentre la parola chiave *limits* indica la quantità massima di risorse che i pod non possono eccedere.
Se la creazione o l'aggiornamento di un oggetto sul cluster viola uno dei vincoli definiti dalle Resource Quota, l'azione fallirà, e Kubernetes fornisce un messaggio che illustra quale vincolo è stato violato e in che modalità.

6.2 Situazione precedente

I nodi del cluster Kubernetes su cui è in esecuzione il servizio di CrownLabs possono fornire un totale di risorse pari a:

- 344 CPU cores
- 2TB di RAM
- circa 25TB di storage in SSD
- circa 10TB di storage in HDD

In questo contesto, per evitare di arrivare alla saturazione delle risorse è stato adottato un modello a limite fissato per ciascun utente.
Nello specifico un utente può creare più istanze in contemporanea fino a quando non raggiunge il suo limite, in caso contrario dovrà cancellare istanze esisenti per crearne di nuove.
Questo modello di assegnazione è stato messo in pratica impostando una Resource Quota legata al namespace di ogni tenant, nello specifico i vincoli imposti sono:

- **requests.cpu** = 10 CPU cores
- **requests.memory** = 25GB
- **limits.cpu** = 15 CPU cores
- **limits.memory** = 25GB

- **count/instances.crownlabs.polito.it** = 5

Tuttavia questo modello non è del tutto ottimale in quanto gli utenti potrebbero avere la necessità di sfruttare la potenza del cluster per tempistiche limitate.
Il lavoro svolto ha quindi avuto come obiettivo quello di fornire una nuova e più flessibile modalità di gestione delle risorse, ispirata al modello della piattaforma di Amazon Elastic Compute Cloud (**Amazon EC2**).
Agli utenti viene fornita una certa quantità di "token" validi per allocare e usare risorse (CPU e RAM) insieme a un sistema di monitoraggio delle risorse attualmente in uso.

6.3 Resource quota per Workspace

Questa sezione illustra il primo passo effettuato nell'ottica di fornire un modello pay-per-use di gestione delle risorse.
L'idea di questo approccio è quella di rendere customizzabili i vincoli imposti dalla Resource Quota per ogni tenant, ed elencati nella sezione precedente.
Nello specifico sono state modificate le CRD delle risorse **Tenant** e **Workspace** con l'aggiunta di un opportuno field contenente le quantità di risorse assegnate. Nel caso del Tenant questo field è duplicato tra Spec e Status, il motivo di questa scelta è illustrato più avanti.
Dopodichè è stata implementata una logica per associare ad ogni workspace una quantità di risorse modificabile dall'amministratore del cluster. Le risorse Tenant e Workspace sono gestite dal **Tenant Operator**, il cui **Controller** si occupa di riconciliare lo stato corrente del Tenant (ciò che è contenuto nello Status) con lo stato desiderato (contenuto nella Spec). La logica per associare le quantità di risorse alle workspace è stata quindi aggiunta all'interno della funzione principale del Tenant Controller, e il suo scopo è quello di calcolare la somma delle risorse per ogni workspace a cui un tenant è iscritto (può essere una o anche più workspace) e aggiornare l'opportuno field nello stato del Tenant.
Se dovesse essere settato il field duale all'interno dello Spec del Tenant, questo andrà a sovrascrivere quello dello Status, ciò servirà per poter customizzare le quantità di risorse anche per il singolo utente.
È stato inoltre configurato un limite delle quantità di risorse totali che possono essere assegnate a un tenant, limite più alto di quello esistente in precedenza e che consiste in:

- 25 CPU cores
- 50GB di RAM
- 10 istanze

Se sommando le quantità di queste tipologie di risorse assegnate a una workspace si ottiene un valore maggiore di quelli appena elencati, esso sarà scartato, e al suo posto sarà usato l'opportuno limite, questo serve per evitare sovrabbondanza nelle risorse assegnate a un singolo utente.
Il seguente pseudocodice illustra la logica aggiunta nel Tenant Controller per gestire l'assegnazione delle risorse, e fornisce una sintesi del codice Go disponibile sul repository GitHub di CrownLabs [21].

```
1: tenant is being reconciled;
2: create a new resourceQuota object;
3: if(tenant.spec.quota is defined)
    4: tenant.status.quota = tenant.spec.quota;
    5: resourceQuota.spec = tenant.spec.quota;
6: else
    7: foreach(workspace in tenant.workspaces)
        8: quota.cpu = quota.cpu + workspace.resourceQuota.cpu;
        9: quota.memory=quota.memory + workspace.resourceQuota.memory;
       10: quota.instances=quota.instances
           + workspace.resourceQuota.instances;
   11: if(quota.cpu >= cpuMax) //currently 25 cores
       12: quota.cpu = cpuMax;
   13: if(quota.memory >= memoryMax) //currently 50 gigabyte
       14: quota.memory = memoryMax;
   15: if(quota.instances >= instancesMax) //currently 10 instances
       16: quota.instances = instancesMax;
   17: tenant.status.quota = quota;
   18: resourceQuota.spec = quota;
19: bind resourceQuota.spec to tenant's namespace;
```

Esempio di esecuzione Per fornire un esempio di esecuzione di questa logica supponiamo di creare nel sistema di CrownLabs il Tenant associato a **Pietro Claudio Lagreca**, con matricola **267549**.
Una volta creato il tenant l'operatore provvede a creare un namespace associato, il cui nome sarà **tenant-s267549**.
Immaginiamo ora di iscrivere questo tenant a due workspace: **Netgroup** e **Cloud Computing**, e che queste workspace abbiano i rispettivi valori nei loro field **spec.quota**:

- 10 cores e 8 cores per la CPU
- 15GB e 25GB per la RAM
- 3 e 5 per il numero massimo di istanze

La logica di calcolo per la Resource Quota totale associata al namespace del tenant produrrà un oggetto che avrà un aspetto simile all'esempio in YAML proposto di seguito:

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: crownlabs-resource-quota
  namespace: tenant-s267549
spec:
  hard:
    requests.cpu: "18"
    requests.memory: 40Gi
    limits.cpu: "18"
    limits.memory: 40Gi
    count/instances.crownlabs.polito.it: 8
```

Listing 6.1: Resource Quota per il tenant s267549

6.3.1 Segnalazione degli errori all'utente

É stata inoltre implementata la logica che permette di avvisare l'utente nel caso in cui sfora il suo limite di risorse con la creazione di una nuova istanza, per risolvere la situazione precedente in cui l'utente attraverso la dashboard di CrownLabs vedeva la sua istanza in perenne stato Starting senza conoscerne il motivo.
Quando l'utente crea una nuova istanza l'Instance Operator provvede ad effettuare la procedura di creazione del rispettivo oggetto nel namespace associato all'utente, ma nel caso in cui l'istanza supera i limiti di risorse, Kubernetes provvede ad inserire un messaggio di errore in un opportuno attributo dell'oggetto

relativo al container o macchina virtuale creata dall'utente, perciò la logica implementata consente di rintracciare questo errore e inserire un messaggio di warning nell'interfaccia grafica dell'utente.

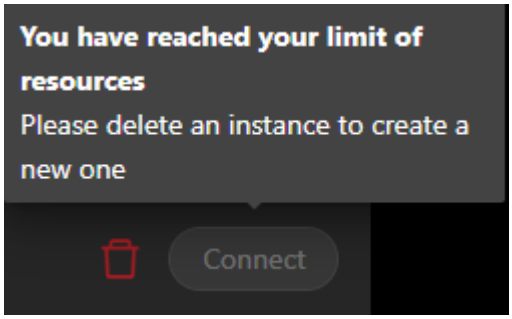

Figura 6.1: Avviso di superamento dei limiti di risorse

Il seguente pseudocodice fornisce una sintesi del codice Go scritto per l'implementazione.

```
1: user creates an instance;
2: instance is starting; //kubernetes checks if there are sufficient resources
3: if(instance.status.conditions.contains("some errors related to insufficient resources"))
    4: return new Phase("resource quota exceeded");
    5: printMessageOnDashboard("Delete some instances to create new ones");
6: else
    7: complete instance creation;
```

6.4 Modello pay-per-use

A questo punto si è arrivati ad una situazione dove esiste ancora un limite alla quantità di risorse che un utente CrownLabs può allocare contemporaneamente, ma non è più un valore fisso, bensì modificabile in base alle necessità dell'utente o alle disponibilità offerte dal cluster.

Lo sviluppo successivo verso un approccio pay-per-use consiste, in via provvisoria, nel monitorare esclusivamente il consumo di CPU dell'utente e legare quest'ultimo ai token assegnati.
Kubernetes nativamente non fornisce oggetti che permettono di effettuare questo tipo di conteggio, perciò il concetto di token è stato introdotto utilizzando risorse custom, così come è stato fatto con l'introduzione dei concetti di Tenant e Workspace.
Questo è stato realizzato tramite l'aggiunta nella CRD del Tenant di un ulteriore field nella sua Spec che contiene le informazioni relative al consumo di token dell'utente.
Il field è formato da un campo contenente il numero di token consumati e un secondo campo che contiene il numero totale di token riservati, i token rappresentano il numero di secondi in cui l'utente ha consumato CPU allocando istanze sul suo namespace.
È stata definita una granularità di 5 minuti, ed è stato configurato un task periodico che ogni 5 minuti calcola il totale di CPU cores delle istanze allocate dall'utente e somma questo totale (moltiplicato per la granularità definita) a ciò che è contenuto nel field dello Status del tenant.
Più aumenta la quantità di tempo in cui l'utente consuma CPU, più diminuiscono i suoi token. I token non si acquistano con denaro reale, ma rappresentano una sorta di valuta virtuale, e fornita periodicamente in una quantità sufficientemente alta, con lo scopo di dare l'idea all'utente che le risorse che lui alloca e usa sono consumate da qualche parte in remoto, e perciò non sono illimitate.
Una volta che l'utente termina i suoi token idealmente gli si impedisce di utilizzare le sue istanze attive e di crearne di nuove, questo è fattibile forzando a 0 la quantità massima di CPU e RAM allocabili, e il numero massimo di istanze che l'utente può creare andando a modificare la Resource Quota del namespace associato al suo Tenant.

6.4.1 Implementazione del conteggio di risorse usate

Questa logica è implementata all'interno di un secondo controller agganciato al **Tenant Operator**.
La reconcile di questo controller è schedulata per essere eseguita ogni 5 minuti, e durante la sua esecuzione itera su tutti i tenant esistenti in CrownLabs, e calcola i secondi di utilizzo di ogni sua istanza seguendo questo comportamento:

- Se l'istanza è stata creata da 5 o più minuti il controller aggiorna l'opportuno field dell'oggetto Tenant considerando l'incremento di 5 minuti, e imposta un ulteriore field di tipo timestamp per tenere traccia di questa modifica.
  Questo è utile per considerare i casi in cui la reconcile dovesse essere stata

innescata dopo un intervallo di tempo diverso dai 5 minuti prestabiliti rispetto al precedente ciclo di reconcile.

- Se l'istanza è stata creata da meno di 5 minuti considera il timestamp di creazione dell'istanza stessa per avere il minutaggio effettivo.

Una volta calcolati i minuti di utilizzo delle istanze i token consumati dal tenant vengono incrementati di una quantità pari ai secondi di utilizzo moltiplicati per il numero di CPU core allocati da ciascuna istanza.
Il seguente box di pseudocodice fornisce una sintesi dell'implementazione in linguaggio Go della logica della reconcile appena descritta

```
1: retrieve tenant from context;
2: foreach(instance in tenant.instances)
    3: retrieve instance template from context;
    4: if(instance.creationTimestamp >
       tenant.spec.token.lastUpdatedTimestamp)
        5: newSeconds = currentTimestamp - instance.creationTimestamp;
        6: //in this case the instance has been created less than five minutes
           before this Reconcile call
    7: else
        8: newSeconds = currentTimestamp
           - tenant.spec.token.lastUpdatedTimestamp;
    9: tenant.spec.token.used += newSeconds*template.numberOfCPUCores;
10: tenant.spec.token.lastUpdatedTimestamp = currentTimestamp;
11: if(tenant.spec.token.used >= tenant.spec.token.reserved)
    12: stop all running instances for this tenant;
    13: prevent tenant from new instance creation;
14: trigger new Reconcile call after 5 minutes;
```

Questo approccio appena descritto non è chiaramente da considerare definitivo, sotto alcuni aspetti si può considerare una traccia di partenza da sviluppare ulteriormente per poter creare un meccanismo completo di controllo delle risorse

utilizzate da ogni utente. Ad esempio in questa configurazione si considera esclusivamente l'utilizzo dei core della CPU, ma all'idea di token bisognerà associare anche l'allocazione e il consumo delle altre tipologie di risorse, quali memoria RAM e memoria di storage.
C'è poi un discorso legato all'esecuzione concorrente di due Controller separati legati allo stesso oggetto, ovvero il Tenant, questo può portare ad una situazione in cui il controller legato al monitoraggio di risorse vada ad innescare la Reconcile del Tenant Controller originale, il quale effettua una quantità di operazioni ben più alta, includendo l'interazione con componenti esterni tipo Keycloak.
Un suggerimento per evitare questo comportamento può essere quello di ridurre la periodicità di esecuzione del controller secondario, aumentando l'intervallo di tempo tra una Reconcile e l'altra da 5 minuti a un nuovo valore dell'ordine di qualche ora, fino a poter magari inglobare la logica di questo controller all'interno di quella del Tenant Controller già esistente, questo eliminerebbe il rischio di conflitti di due controller che operano sullo stesso oggetto.

Capitolo 7

Conclusioni

Questa attività di tesi ha voluto presentare un'esplorazione del mondo dei servizi Cloud, e su come questi sono resi disponibili all'utente sfruttando l'orchestrazione e la gestione dei container di Kubernetes.
É stato fornito l'esempio relativo al caso d'uso di Faveo Helpdesk, un prodotto già pronto all'uso in modalità On-premise, ma privo di alcuni componenti necessari per essere rilasciato in un cluster Kubernetes.
Ad oggi Faveo è correttamente integrato nel contesto di CrownLabs e quindi comodamente disponibile ai suoi utenti per l'uso, a questa comodità si aggiungono tutti i benefici che può garantire questo tipo di soluzione per gestire le comunicazioni con gli utenti tramite ticket.
É presente un discreto numero di utenti registrati su Faveo, ad oggi poco più di 30, il che fornisce già un primo feedback a riguardo della solidità del sistema.
Altro aspetto affrontato nell'attività di tesi ha riguardato la gestione delle quantità di risorse computazionali fornite agli utenti, che ha permesso di migrare dal precedente modello a limiti fissati ad un modello dove si cerca comunque di evitare l'esaurimento delle risorse disponibili sul cluster, ma facendo in modo che l'utente possa sfruttare in modo più elastico la potenza del Cloud fornito, purchè rientri nei limiti imposti periodicamente dal meccanismo dei token.

7.1 Sviluppi futuri

In aggiunta a quanto affermato alla fine del precedente capitolo, un ulteriore spunto per rendere il modello pay-per-use più completo può essere rappresentato dalla differenziazione tra l'allocazione e l'effettivo consumo di risorse.
La logica implementata considera solo la creazione delle istanze, ma non analizza l'effettivo utilizzo che l'utente fa di queste istanze, queste informazioni possono essere rese disponibili attraverso le metriche esposte attraverso Prometheus, che

analizzano le prestazioni globali del cluster CrownLabs, ma anche quelle delle singole istanze virtuali usate dagli utenti della piattaforma.
Una volta ottenute queste informazioni sarà possibile sia fornire all'utente alcune dashboard tramite Grafana che possano schematizzare il suo complessivo utilizzo di risorse in funzione dei suoi token disponibili nel corso del tempo, sia differenziare il consumo dei token stessi a seconda che l'utente allochi le istanze senza utilizzarle (in una configurazione a basso consumo) o che le allochi utilizzandole (consumo più alto). Per quanto riguarda invece le istanze persistenti fornite da CrownLabs, può essere verosimile arrestare il conteggio dei token nel momento in cui l'utente mette in pausa la sua istanza, perchè di fatto in questa configurazione non c'è consumo di risorse computazionali.
Le istanze persistenti possono essere paragonate a veri e propri computer fisici, il che significa che metterle in pausa è paragonabile all'attività di arresto del sistema, considerando ciò ha senso quindi che il conteggio dei token sia arrestato quando si verifica questo tipo di situazione.
La presentazione di questa attività termina quì, in queste ultime righe colgo l'occasione per esprimere un ringraziamento al Professore Fulvio Risso, a Marco Iorio, a Federico Cucinella e tutto il team CrownLabs per avermi dato la possibilità di collaborare con loro e il grande supporto fornito in questi mesi.

Bibliografia

[1] Bean, Martin. Laravel 5 essentials. Packt Publishing Ltd, 2015.

[2] Dobies, Jason, and Joshua Wood. Kubernetes Operators: Automating the Container Orchestration Platform. O'Reilly Media, 2020.

[3] Deployment https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

[4] Docker https://www.docker.com/

[5] Faveo Helpdesk https://www.faveohelpdesk.com/

[6] Get started with Golang Language. https://go.dev/

[7] Git https://git-scm.com/

[8] Hayut, Yehuda. "Containerization and the load center concept." Economic geography 57.2 (1981): 160-176.

[9] Huang, Zhuo, et al. "Fastbuild: Accelerating docker image building for efficient development and deployment of container." 2019 35th Symposium on Mass Storage Systems and Technologies (MSST). IEEE, 2019.

[10] Kinsta. Cosa è GitHub https://kinsta.com/it/knowledgebase/cosa-e-github/

[11] Kubernetes. Cos'è Kubernetes. 2020 https://kubernetes.io/it/docs/concepts/overview/what-is-kubernetes/

[12] Kubernetes Resource Quota https://kubernetes.io/docs/concepts/policy/resource-quotas

[13] Politecnico Di Torino. About CrownLabs. 2020 https://crownlabs.polito.it/about/

[14] Red Hat. Cosa sono i Servizi Cloud. 2019 https://www.redhat.com/it/topics/cloud-computing/what-are-cloud-services

[15] Keycloak https://www.keycloak.org/

[16] Operatore MySQL sviluppato da Bitpoke https://github.com/bitpoke/mysql-operator

[17] Radha, V., and D. Hitha Reddy. "A survey on single sign-on techniques." Procedia Technology 4 (2012): 134-139.

[18] Rahman, Mohammad Masudur, and Chanchal K. Roy. "An insight into the pull requests of github." Proceedings of the 11th working conference on mining software repositories. 2014.

[19] Red Hat OpenShift. High Availability Overview https://docs.openshift.com/container-platform/3.11/admin_guide/high_availability.html

[20] ReplicaSet https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

[21] Repository GitHub di CrownLabs https://github.com/netgroup-polito/CrownLabs

[22] Repository GitHub di Faveo Helpdesk di Netgroup Polito https://github.com/netgroup-polito/faveo-helpdesk

[23] Repository GitHub originale di Faveo Helpdesk di Ladybird Web Solution https://github.com/ladybirdweb/faveo-helpdesk

[24] StatefulSet https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/