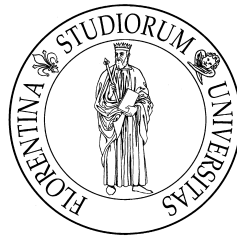


UNIVERSITÀ DEGLI STUDI DI FIRENZE  
Facoltà di Scienze Matematiche, Fisiche e Naturali  
Corso di Laurea Magistrale in Informatica



Tesi di Laurea

IL PROGETTO INDICO KT  
MIGLIORARE L'IMPATTO A LIVELLO MONDIALE DI INDICO

TOMMASO PAPINI

Relatore: *Pierluigi Crescenzi*

Anno Accademico 2014-2015



## ABSTRACT

---

Nel 2014 il CERN (Conseil Européen pour la Recherche Nucléaire) ha celebrato i suoi *60 anni* di attività.

Dalla sua fondazione, nel lontano Settembre 1954, il CERN è stato protagonista di una serie di scoperte e innovazioni in svariati settori scientifici, informatica e fisica in primis, e si è affermato come uno tra i più importanti centri di ricerca in Europa e nel mondo. Il CERN è infatti il più grande centro di ricerca di fisica nucleare in Europa, possessore, al momento, del più grande acceleratore di particelle al mondo: l'LHC (Large Hadron Collider).

Grazie al CERN sono state fatte molte scoperte in ambito della fisica delle particelle, come la recente conferma sperimentale dell'effettiva esistenza del tanto cercato *bosone di Higgs*: il 4 Luglio 2012 i team degli esperimenti ATLAS (A Toroidal LHC ApparatuS) e CMS (Compact Muon Solenoid) confermarono la scoperta del bosone di Higgs, scoperta che portò poi, il 10 Dicembre 2013, al conferimento del premio Nobel per la fisica a Peter Higgs e François Englert, principali ricercatori che teorizzarono l'esistenza di questa particella subatomica.

Ma il CERN è stato protagonista anche di molte innovazioni in ambito informatico: pensiamo al servizio internet Web, o WWW (World Wide Web), che fu ideato proprio al CERN nel 1989 da Tim Berners-Lee, adesso fondatore e presidente del W3C (World Wide Web Consortium).

È in questo contesto che si inquadra il progetto *Improving the Worldwide Impact of Indico* che il sottoscritto, Tommaso Papini, ha seguito nel corso dei 14 mesi passati al CERN con il programma di *Technical Student* (dal 1 Ottobre 2013 al 30 Novembre 2014), sotto la supervisione di Pedro Ferreira (dipendente del CERN ed attuale Project Manager di Indico) al CERN e del Prof. Pierluigi Crescenzi (professore presso l'Università degli Studi di Firenze) da Firenze.

Quest'elaborato, volto ad essere un resoconto di questi 14 mesi, sarà suddiviso in tre parti ed esporrà, in modo comprensivo ed esaustivo, il progetto seguito in Indico. All'interno della prima parte verrà esposta una panoramica dell'ambiente CERN e del progetto Indico. Nella seconda parte, invece, saranno esposte le varie parti e le varie fasi che hanno composto il progetto stesso nell'arco di questi 14 mesi, assieme ad una panoramica sugli strumenti ed i linguaggi utilizzati. Infine, nella parte conclusiva, si parlerà di altri progetti minori seguiti al CERN e si esporranno delle note conclusive del progetto seguito.

## INDICE

---

<b>I</b>	<b>CERN E INDICO</b>	<b>1</b>
1	CERN	2
1.1	Breve storia . . . . .	3
1.1.1	Principali scoperte scientifiche . . . . .	3
1.1.2	Il CERN e l'informatica . . . . .	5
1.2	Il complesso degli acceleratori . . . . .	6
1.2.1	Large Hadron Collider . . . . .	8
1.3	Lavoro al CERN . . . . .	9
1.3.1	Cultura e valori . . . . .	9
1.3.2	Carriera al CERN . . . . .	10
1.3.3	La struttura generale . . . . .	11
1.3.4	Knowledge Transfer . . . . .	12
2	INDICO	13
2.1	Principali caratteristiche . . . . .	13
2.2	Dettagli tecnici . . . . .	16
2.3	La community . . . . .	17
<b>II</b>	<b>INDICO KT PROJECT</b>	<b>19</b>
3	KT PROJECT: UNA PANORAMICA	20
3.1	Progetti principali . . . . .	20
3.1.1	Cloud Deployment . . . . .	21
3.1.2	Distribuzione e Packaging . . . . .	21
3.1.3	Instance Tracker . . . . .	21
3.1.4	Conference Customization Prototype . . . . .	21
3.2	Strumenti e linguaggi . . . . .	21
3.2.1	Github . . . . .	21
3.2.2	Python . . . . .	21
3.2.3	Cloud-init . . . . .	22
3.2.4	Fabric . . . . .	23
3.2.5	QEMU e KVM . . . . .	24
3.2.6	Virtualenv . . . . .	25
3.2.7	Requests . . . . .	25
3.2.8	Twitter Bootstrap . . . . .	25
3.2.9	Flask . . . . .	25
3.2.10	Celery . . . . .	25
3.2.11	PostgreSQL e SQLAlchemy . . . . .	25
3.2.12	Jinja2 . . . . .	25
3.2.13	jqPlot . . . . .	25

3.2.14	jVectorMap . . . . .	25
4	CLOUD DEPLOYMENT . . . . .	26
4.1	Deployment con cloud-init . . . . .	28
4.1.1	Fase di configurazione . . . . .	29
4.1.2	Generazione dello script bash . . . . .	31
4.1.3	Generazione del file cloud-config . . . . .	32
4.1.4	Generazione del file <code>user-data</code> . . . . .	34
4.2	Creazione di immagini virtuali . . . . .	34
4.2.1	Avvio della macchina virtuale . . . . .	35
4.2.2	Configurazione della macchina virtuale . . . . .	37
4.3	Script di gestione remota . . . . .	38
5	DISTRIBUZIONE E PACKAGING . . . . .	40
5.1	Generazione di una distribuzione . . . . .	41
5.2	Upload della distribuzione . . . . .	43
5.2.1	Upload su Github . . . . .	43
5.2.2	Upload su un server . . . . .	44
6	INSTANCE TRACKER . . . . .	46
6.1	Caratteristiche principali . . . . .	48
6.1.1	Installazione e configurazione . . . . .	48
6.1.2	Non solo Indico . . . . .	50
6.1.3	Campi principali e campi estratti . . . . .	51
6.1.4	Interfaccia web . . . . .	54
6.1.5	Scheduler . . . . .	59
6.1.6	Script di gestione . . . . .	61
6.2	Dettagli implementativi . . . . .	62
6.2.1	Struttura del codice . . . . .	62
6.2.2	Il Database . . . . .	62
6.2.3	Endpoint ed API . . . . .	64
6.2.4	Interfaccia web e applicazione . . . . .	66
6.2.5	Javascript . . . . .	68
6.2.6	Estrazione dei dati . . . . .	69
6.3	Adattamento di Indico . . . . .	71
6.3.1	Prima configurazione . . . . .	72
7	CONFERENCE CUSTOMIZATION PROTOTYPE . . . . .	75
8	ALTRI PROGETTI . . . . .	76
8.1	I ticket . . . . .	76
8.2	Nuova pagina delle statistiche . . . . .	76
8.3	Installazione su Windows Server . . . . .	76
	<b>III NOTE E CONCLUSIONI . . . . .</b>	<b>77</b>
9	CONCLUSIONI . . . . .	78
	<b>BIBLIOGRAFIA . . . . .</b>	<b>79</b>

## ELENCO DELLE FIGURE

---

Figura 1	Logo del CERN . . . . .	2
Figura 2	Corrente debole neutra . . . . .	4
Figura 3	Cavità RF del LEP . . . . .	4
Figura 4	Server Room del Data Center . . . . .	6
Figura 5	Complesso degli acceleratori . . . . .	7
Figura 6	Visita all’LHC . . . . .	9
Figura 7	Organigramma del CERN . . . . .	12
Figura 8	Logo di Indico . . . . .	13
Figura 9	Timetable in Indico (esempio) . . . . .	14
Figura 10	Categoria in Indico (esempio) . . . . .	15
Figura 11	Meeting in Indico (esempio) . . . . .	15
Figura 12	Linguaggi di Indico . . . . .	16
Figura 13	Logo di Fabric . . . . .	23
Figura 14	Asset della release 1.2 . . . . .	44
Figura 15	Stilizzazione di Cephalopod . . . . .	47
Figura 16	Barra del menu e breadcrumbs . . . . .	55
Figura 17	Lista delle istanze . . . . .	56
Figura 18	Lista delle istanze (con menu a tendina aperto) . . . . .	57
Figura 19	Dettagli di un’istanza . . . . .	58
Figura 20	Statistiche principali . . . . .	59
Figura 21	Statistiche definite dall’amministratore . . . . .	60
Figura 22	Configurazione di Cephalopod in Indico . . . . .	72
Figura 23	Avviso della mancata sincronia con Cephalopod . . . . .	73
Figura 24	Prima configurazione di Indico . . . . .	74

## LISTATO DEL CODICE

---

Codice 1	Comando <code>.format()</code> (esempio) . . . . .	22
Codice 2	Boot con cloud-init (esempio OpenStack) . . . . .	22
Codice 3	Esecuzione task fabric (esempio) . . . . .	24
Codice 4	Configurazione cloud-init . . . . .	30
Codice 5	Configurazione automatica di Indico . . . . .	32
Codice 6	Abilitazione di <code>sudo</code> per script cloud-init . . . . .	32
Codice 7	Struttura del template per cloud-config . . . . .	33

Codice 8	Comando per la generazione del file MIME multiparti . . . .	34
Codice 9	Avvio della macchina virtuale . . . . .	36
Codice 10	Comando di attesa della terminazione del boot . . . . .	37
Codice 11	Comando <code>put()</code> per link simbolici . . . . .	38
Codice 12	Selezione della versione di Indico per la distribuzione . . . .	42
Codice 13	Configurazione per l'upload a Github (esempio) . . . . .	43
Codice 14	API per l'upload a Github . . . . .	44
Codice 15	Configurazione per l'upload su server (esempio) . . . . .	45
Codice 16	<code>clone</code> del codice di Cephalopod . . . . .	48
Codice 17	Installazione di Cephalopod (virtualenv) . . . . .	48
Codice 18	Installazione di Cephalopod (globale) . . . . .	48
Codice 19	Copia delle configurazioni di Cephalopod . . . . .	49
Codice 20	Configurazioni d'esempio di Cephalopod . . . . .	50
Codice 21	Esempio di campo aggiuntivo ( <code>python_version</code> ) . . . . .	53
Codice 22	Esempio di campo aggiuntivo ( <code>events</code> ) . . . . .	54
Codice 23	Impostazioni di Celery (esempio) . . . . .	60
Codice 24	Script di gestione (con <code>setup.py</code> ) . . . . .	61
Codice 25	Script di gestione . . . . .	61
Codice 26	Definizione della classe <code>User</code> . . . . .	63
Codice 27	Definizione di una classe JSON per SQLAlchemy . . . . .	64
Codice 28	<code>crawled_data</code> e <code>geolocation</code> . . . . .	64
Codice 29	Funzioni per la lista delle istanze e eliminazione di un'istanza	67
Codice 30	Grafico a torta degli stati . . . . .	69
Codice 31	Elemento HTML del grafico a torta . . . . .	69
Codice 32	API di geolocalizzazione . . . . .	70

## ACRONIMI

---

ACM	Association for Computing Machinery
AD	Antiproton Decelerator
ALICE	A Large Ion Collider Experiment
API	Application Programming Interface
ATLAS	A Toroidal LHC ApparatuS
AVC	AudioVisual and Collaborative Services
BE	Beams

CERN	Conseil Européen pour la Recherche Nucléaire
CIS	Collaboration & Information Services
CMS	Compact Muon Solenoid
COMPASS	Common Muon and Proton Apparatus for Structure and Spectroscopy
CP	Charge Parity
CSS	Cascading Style Sheets
CSV	Comma Separated Values
DB	Database
DG	Director General
EGEE	Enabling Grids for E-sciencE
EN	Engineering
FNAL	Fermi National Accelerator Laboratory
FP	Finance, Procurement and Knowledge Transfer
GS	General Infrastructure Services
HEP	High Energy Physics
HR	Human Resources
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IC	Indefinite Contract
IHEP	Institute of High Energy Physics
ILC	International Linear Collider
INFN	Istituto Nazionale di Fisica Nucleare
IP	Internet Protocol
ISOLDE	On-Line Isotope Mass Separator
IT	Information Technology
JSON	JavaScript Object Notation



KT	Knowledge Transfer
KVM	Kernel-based Virtual Machine
LEAR	Low Energy Antiproton Ring
LEIR	Low Energy Ion Ring
LEP	Large Electron–Positron Collider
LHC	Large Hadron Collider
LHCb	Large Hadron Collider beauty
LHCf	Large Hadron Collider forward
Linac	Linear Accelerator
MIME	Multipurpose Internet Mail Extensions
MoEDAL	Monopole and Exotics Detector at the Large Hadron Collider
OERN	Organisation Européenne pour la Recherche Nucléaire
ORDBMS	Object-Relational Database Management System
PH	Physics
PS	Proton Synchrotron
PSB	Proton Synchrotron Booster
QEMU	Quick EMUlator
REST	Representational State Transfer
RHEL	Red Hat Enterprise Linux
SL6	Scientific Linux 6
SLC6	Scientific Linux CERN 6
SMTP	Simple Mail Transfer Protocol
SPS	Super Proton Synchrotron
SSH	Secure Shell
SSL	Secure Sockets Layer
TE	Technology
TOTEM	TOTal Elastic and diffractive cross section Measurement

TTY	Teletypewriter
UI	User Interface
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
VM	Virtual Machine
W <sub>3</sub> C	World Wide Web Consortium
WSGI	Web Server Gateway Interface
WYSIWYG	What You See Is What You Get
WWW	World Wide Web
XML	eXtensible Markup Language
ZODB	Zope Object Database

Parte I

CERN E INDICO

## CERN

---

Il CERN, anche noto come Organizzazione Europea per la Ricerca sul Nucleare, è l'organizzazione che opera il più grande laboratorio di fisica delle particelle al mondo.

Venne istituito il 29 Settembre 1954 da una serie di stati fondatori nei pressi di Ginevra, a cavallo del confine franco-svizzero. Al momento, il CERN vanta un totale di 21 stati membri, dei quali soltanto Israele non europeo.

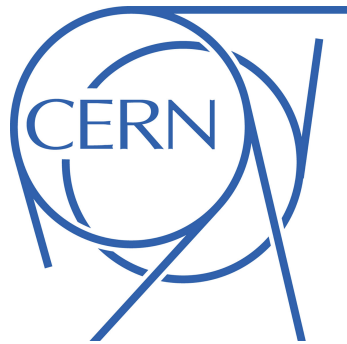


Figura 1: Logo del CERN.

Col termine CERN si indica spesso il laboratorio stesso che, al 2013, contava 2'513 membri staff e circa 12'313 tra fellow, collaboratori associati e studenti, rappresentando un totale di 608 università e 113 nazionalità.

L'obiettivo principale del CERN è quello di fornire acceleratori di particelle ed altri strumenti e infrastrutture necessarie per le ricerche di fisica delle alte energie.

Al CERN è stata anche proposta e sviluppata una prima implementazione del World Wide Web. All'interno del sito di Meyrin del CERN è presente il Data Center principale, che vanta una serie di infrastrutture per l'elaborazione di dati, con lo scopo principale di analizzare dati sperimentali provenienti dal complesso degli acceleratori. Per questo motivo, e per il fatto che questi dati dovevano essere disponibili ed analizzabili da molti altri centri di ricerca, spesso molto distanti, il Data Center di Meyrin divenne un grandissimo centro di smistamento dati, formando una vasta rete tra i vari centri di ricerca.

## 1.1 BREVE STORIA

La convenzione che istituì il CERN venne ratificata il 29 Settembre 1954 da 12 stati dell'Europa occidentale.

L'acronimo CERN, in origine, indicava la sigla francese *Conseil Européen pour la Recherche Nucléaire* (Consiglio Europeo per la Ricerca sul Nucleare), che rappresentava il consiglio provvisorio istituito nel 1952 dai 12 stati fondatori per la costruzione di un laboratorio di ricerca di fisica nucleare europeo. Quando il consiglio venne sciolto nel 1954 il nome cambiò all'attuale Organizzazione Europea per la Ricerca sul Nucleare (in francese, *Organisation Européenne pour la Recherche Nucléaire*). In quest'occasione si decise di mantenere la vecchia sigla CERN al posto della, più brutta e meno d'impatto, sigla OERN. Il fisico Heisenberg stesso si pronunciò sul fatto che l'acronimo sarebbe dovuto rimanere CERN anche se il nome non lo era più.

Il CERN viene considerato un laboratorio di pace in quanto grazie ad esso persone provenienti da tutto il mondo hanno la possibilità di incontrarsi, discutere e collaborare su diversi progetti. Grazie al CERN riescono a lavorare assieme anche persone provenienti da stati in guerra tra loro, come ad esempio Israele e Palestina.

Inoltre, il CERN venne istituito circa 10 anni dopo la costruzione della bomba atomica come conseguenza principale della II guerra mondiale.

### 1.1.1 Principali scoperte scientifiche

In origine, lo scopo principale del CERN era quello di studiare i nuclei atomici, ma ben presto l'attenzione si volse verso lo studio della fisica delle alte energie, in particolare all'interazione tra le particelle subatomiche.

1973 Nel 1973 venne scoperta, grazie alla camera a bolle Gargamelle, una particolare interazione tra particelle subatomiche, detta *corrente debole neutra* (Figura 2) che, nel 1979, permise l'attribuzione del Premio Nobel per la Fisica ad Abdus Salam, Sheldon Glashow e Steven Weinberg, che teorizzarono quest'interazione in passato.

1983 Nel 1983 venne fatta un'altra importantissima scoperta di fisica delle particelle, ovvero la scoperta dei bosoni W e Z. La teorizzazione di questi bosoni fu una conseguenza quasi immediata dell'osservazione della corrente debole neutra, ma prima di poter osservare direttamente queste particelle subatomiche si dovette aspettare di avere un acceleratore di particelle abbastanza potente da produrle. Il primo acceleratore in grado di fare ciò installato al CERN fu il Super Proton Synchrotron (SPS), che permise, nel Gennaio del 1983 di osservare chiare tracce del bosone W. Gli esperimenti principali furono due, condotti in parallelo,

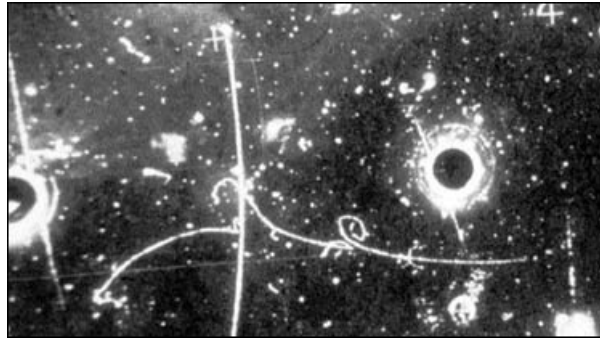


Figura 2: L'evento di corrente debole neutra osservato con Gargamelle.

e vennero chiamati UA1 (condotto da Carlo Rubbia) e UA2 (condotto da Pierre Darriulat). I due esperimenti osservarono anche, nel Maggio 1983, i bosoni Z, grazie all'introduzione della tecnica del raffreddamento stocastico introdotta da Simon van der Meer. Nel 1984, la fondazione Nobel, attribuì a Rubbia e Van der Meer il Premio Nobel per la Fisica per l'insostituibile sforzo messo in queste ricerche.

1989 Nel 1989 venne confermato sperimentalmente, utilizzando il LEP (Large Electron-Positron Collider), che il numero di famiglie di particelle con neutrini leggeri è 3, numero consistente con il Modello Standard.

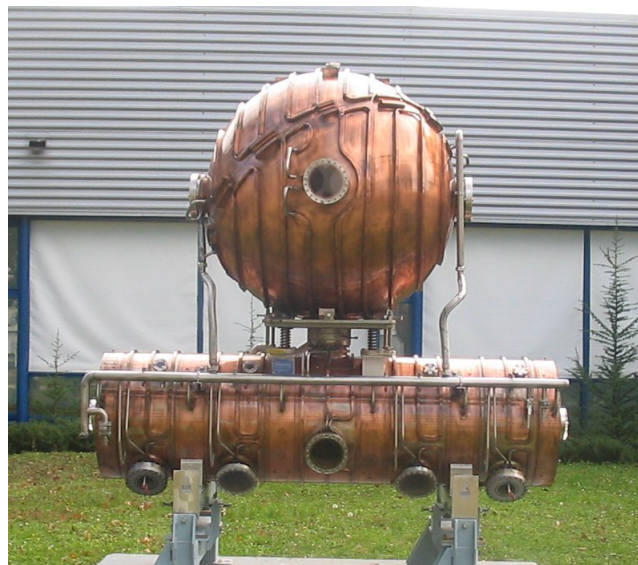


Figura 3: Una cavità RF del LEP esposta al Microcosm del CERN.

1992 Nel 1992 venne conferito il Premio Nobel per la Fisica a Georges Charpak per “per l’invenzione e lo sviluppo dei rivelatori di particelle, in particolare della camera proporzionale a multifili”.

1995 Nel 1995 vennero, per la prima volta nella storia, creati atomi di anti-idrogeno grazie all’esperimento PS210 realizzato al LEAR (Low Energy Antiproton Ring). Questi atomi di anti-idrogeno erano molto instabili e si distrussero subito dopo la creazione, ma non senza produrre un segnale elettrico unico, indice che erano effettivamente stati creati atomi di anti-idrogeno.

1999 L’esperimento NA48 dimostrò, nel 1999, la violazione della simmetria CP (Charge Parity). L’esperimento fu lanciato nel 1990 come successore dell’esperimento NA31 con lo specifico scopo di dimostrare la violazione della simmetria CP, la quale afferma che le leggi della fisica dovrebbero rimanere le stesse quando una particella viene sostituita con la sua anti-particella (simmetria di coniugazione di carica, C) e le coordinate spaziali invertite (simmetria di parità, P).

2010 Nel 2010 un gruppo di ricercatori guidati dal fisico Jeffrey Hangst riuscì ad isolare 38 atomi di anti-idrogeno per circa un decimo di secondo, mentre nel 2011 si riuscì a mantenere degli atomi di anti-idrogeno per più di 15 minuti, in modo da poter studiare più a fondo l’antimateria e le sue proprietà.

2012 Infine, nel 2012, venne finalmente osservato un bosone di massa circa  $125 \text{ GeV}/c^2$  consistente con il tanto cercato bosone di Higgs. Questo bosone venne teorizzato più di 40 anni fa dai ricercatori Peter Higgs e François Englert ed ha una grande importanza per la fisica delle particelle in quanto la sua esistenza può spiegare come mai alcune particelle hanno massa. L’osservazione definitiva del bosone di Higgs avvenne il 4 Luglio 2012, sfruttando le collisioni prodotte dall’LHC, e venne osservato in contemporanea dai due esperimenti CMS e ATLAS. Questa importantissima scoperta portò al conferimento del Premio Nobel per la Fisica, il 10 Dicembre 2013, ai ricercatori Higgs e Englert.

### 1.1.2 Il CERN e l’informatica

Il primo computer arrivò al CERN nel 1959 e da allora i fisici iniziarono ad utilizzare sempre più gli strumenti informatici per le loro analisi. Un elemento importante in questo contesto fu l’italiano Paolo Zanella, capo del dipartimento IT (Information Technology) per 13 anni, dal 1976 al 1989. Gli esperimenti condotti al CERN producevano una mole di dati sempre più grande, tale da rendere impossibile la sola analisi “a mano”.

Nel tempo, venne anche sperimentato il collegamento fra più calcolatori

portando all'utilizzo delle prime reti di calcolatori. Presto, al CERN si formò uno dei più grandi centri di calcolo in Europa. Più recentemente si sono iniziate a sviluppare al CERN tecnologie per il grid computing, con progetti come Enabling Grids for E-sciencE (EGEE) o LHC Computing Grid.



Figura 4: L'interno della Server Room del Data Center nel sito di Meyrin.

Il servizio Internet World Wide Web prese vita al CERN dal progetto ENQUIRE, condotto da Tim Berners-Lee nel 1989 e Robert Cailliau nel 1990. Nel 1995 Berners-Lee e Cailliau vennero premiati dalla ACM (Association for Computing Machinery) per l'indispensabile contributo allo sviluppo del World Wide Web. Il progetto era basato sul concetto di ipertesto, ovvero testo riprodotto su display e collegato ad altri testi tramite iperlink. L'obiettivo originale era quello di facilitare lo scambio di informazioni tra i ricercatori per condurre analisi ed esperimenti.

Il primo sito web fu attivo nel 1991 ed il 30 Aprile 1993 il CERN annunciò pubblicamente che il Web sarebbe stato libero per tutti.

## 1.2 IL COMPLESSO DEGLI ACCELERATORI

Al CERN viene utilizzata una serie di acceleratori, disposti in sequenza in modo da aumentare l'energia delle particelle prima di essere ridirette agli eventuali esperimenti o all'acceleratore successivo. Il logo stesso del CERN (Figura 1) è ispirato al complesso degli acceleratori presente.

I macchinari attualmente attivi al CERN sono i seguenti:

**LINAC2 E LINAC3** Questi due acceleratori lineari (Linac sta per Linear Accelerator) generano particelle a bassa energia. Linac2 accelera protoni fino a 50 MeV



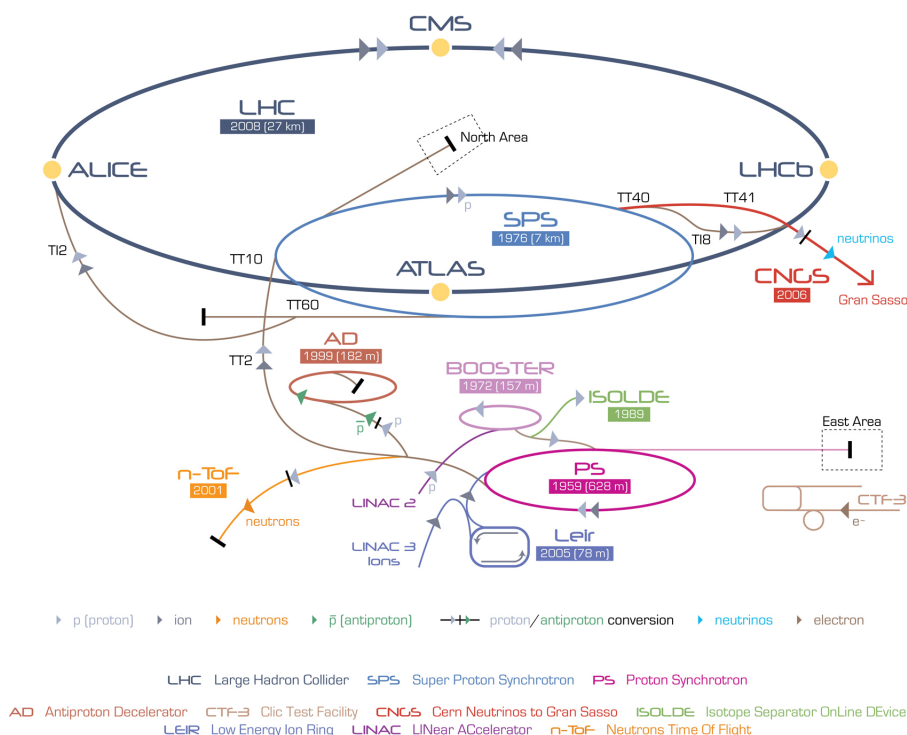


Figura 5: Uno schema del complesso degli acceleratori presenti al CERN con legenda.

per poi iniettarli nel Proton Synchrotron Booster (PSB), mentre Linac3 genera ioni pesanti a  $4.2 \text{ MeV/u}$  per iniettarli nel Low Energy Ion Ring (LEIR).

**PROTON SYNCHROTRON BOOSTER** Aumenta l'energia dei protoni prodotti dal Linac2 prima di trasferirli ad altri acceleratori.

**LOW ENERGY ION RING** Accelera gli ioni pesanti prodotti dal Linac3 prima di trasferirli al Proton Synchrotron (PS). Questo acceleratore fu commissionato nel 2005 come riconfigurazione del precedente acceleratore, il LEAR.

**PROTON SYNCHROTRON** Accelera le particelle fino a 28 GeV. Fu costruito tra il 1954 ed il 1959 ed è ancora attivo come alimentatore del più potente SPS.

**SUPER PROTON SYNCHROTRON** Acceleratore circolare installato in un tunnel di 2 Km di diametro ed attivo dal 1976. Fu costruito inizialmente per portare le particelle fino a 300 GeV, per poi essere gradualmente aggiornato fino ai 450 GeV. Questo acceleratore ha i propri esperimenti, al momento COMPASS (Common Muon and Proton Apparatus for Structure and Spectroscopy) ed NA62, ma veniva anche utilizzato come collisore protoni-antiprotoni e per accelerare elettroni e positroni prima di iniettarli nel LEP. Dal 2008 viene invece

utilizzato per iniettare protoni e ioni pesanti nell'LHC.

**ON-LINE ISOTOPE MASS SEPARATOR** Anche indicato con la sigla ISOLDE, questa struttura viene utilizzata per studiare i nuclei instabili. Ioni radioattivi provenienti dal PSB vengono fatti scontrare con un'energia tra 1.0 e 1.4 GeV.

**ANTIPROTON DECELERATOR** Anche noto come AD, serve a ridurre la velocità degli antiprotoni di circa un 10% della velocità della luce per poter studiare l'antimateria.

**COMPACT LINEAR COLLIDER** Studia la fattibilità di possibili acceleratori lineari futuri.

### 1.2.1 *Large Hadron Collider*

L'LHC è attualmente il più grande acceleratore di particelle installato al CERN, ovvero l'acceleratore principale su cui vengono eseguiti la maggior parte degli esperimenti.

Il tunnel dell'LHC è situato 100 metri sotto terra, nella zona tra l'Aeroporto Internazionale di Ginevra e i monti del Giura. Il tunnel ha una circonferenza di circa 27 Km ed era precedentemente occupato dal vecchio acceleratore LEP, che venne disattivato definitivamente nel 2000. Gli acceleratori PS ed SPS vengono utilizzati per pre-accelerare i protoni iniettati nell'LHC.

Lungo l'acceleratore sono presenti i suoi sette esperimenti attualmente attivi: CMS, ATLAS, LHCb (Large Hadron Collider beauty), MoEDAL (Monopole and Exotics Detector at the Large Hadron Collider), TOTEM (TOTAl Elastic and diffractive cross section Measurement), LHCf (Large Hadron Collider forward) ed ALICE (A Large Ion Collider Experiment).

L'LHC iniziò fin da subito a generare una mole enorme di dati da inviare a diversi laboratori sparsi per il mondo in modo da poter eseguire calcoli distribuiti utilizzando una struttura grid dedicata: la LHC Computing Grid.

Il primo fascio di particelle fu iniettato nell'LHC nell'Agosto del 2008 ma, a causa di problemi tecnici, non venne riattivato fino al Novembre del 2009. Il 30 Marzo 2010, invece, si verificò la prima collisione tra due fasci di protoni, ognuno alla velocità di 3.5 TeV, generando una collisione di 7 TeV. Dal Marzo 2012 si passò invece a collisioni di 8 TeV (ovvero 4 TeV in ogni direzione). Agli inizi del 2013 l'LHC venne disattivato per avviare un periodo di manutenzione e aggiornamento della durata di 2 anni. Dai primi mesi del 2015, l'LHC è stato finalmente rimesso in funzione per sperimentare le prime collisioni a 13 TeV. Nel Luglio 2012 venne annunciata l'osservazione sperimentale di una particella subatomica consistente con il tanto agognato bosone di Higgs. Nel Marzo 2013



(a) ATLAS



(b) CMS

Figura 6: Il sottoscritto in visita ad alcuni esperimenti dell'LHC.

venne confermato dal CERN che, a seguito di una serie di analisi condotte sui dati sperimentali, la particella trovata era effettivamente il bosone di Higgs.

### 1.3 LAVORO AL CERN

Il CERN è il risultato di una collaborazione tra stati, università, scienziati e studenti che sono motivati non da un margine di profitto, ma bensì da un impegno a creare e condividere la conoscenza.

Le persone del CERN, sia che siano impiegati fissi, che soltanto di passaggio, prendono parte a ricerche e collaborazioni interessanti con persone di tutto il mondo, cercando di risolvere i misteri della fisica o di creare qualcosa di nuovo.

#### 1.3.1 *Cultura e valori*

Al CERN vige il Code of Conduct (Codice della Condotta), ovvero una guida su come trattare gli altri colleghi in accordo con i valori del CERN. [2]

I cinque valori di base su cui si basa il CERN sono:

**INTEGRITÀ** Ovvero comportarsi in modo etico, onesto e ritenersi responsabile delle proprie azioni.

**IMPEGNO** Dimostrare, durante il proprio lavoro, un alto livello di motivazione e dedizione verso l'organizzazione ed il proprio progetto.

**PROFESSIONALITÀ** Produrre risultati di alta qualità rispettando i limiti di tempo e le risorse previsti per il completamento del lavoro.

**CREATIVITÀ** Promuovere l'innovazione nel proprio settore di lavoro e permettere lo sviluppo e l'evoluzione dell'organizzazione.

**DIVERSITÀ** Ovvero saper apprezzare le differenze e promuovere l'uguaglianza e la collaborazione.

### 1.3.2 *Carriera al CERN*

Il CERN ammette un ampio ventaglio di possibilità per fare carriera ed acquisire esperienza lavorativa al suo interno. Ci sono svariati tipi di contratti per tutti i tipi di aspiranti dipendenti o collaboratori CERN: contatti di pochi mesi o contratti di anni, contratti per studenti, per laureati o per professionisti, ecc. ... [1]

Uno dei principi di assunzione e lavoro al CERN è la non rinnovabilità del contratto: ogni tipo di contratto può essere assegnato ad ogni individuo al più una sola volta. Questo serve a garantire la diversità ed il continuo ricambio di personale ed allo stesso tempo formare abitanti degli stati membri con un'esperienza di alto livello.

**STUDENTE** In qualità di studente universitario si hanno molte possibilità di lavoro al CERN con contratti di Summer, Admin, Technical e Doctoral Student. Il programma di Technical Student, vissuto dal sottoscritto, prevede una durata dai 6 ai 12 mesi, con la possibilità di estendere il contratto di 2 ulteriori mesi (risorse permettendo). Questi tipi di contratto permettono agli studenti di fare esperienza sul campo ed affrontare sfide intellettuali con alte possibilità di formazione.

**LAUREATO** I contratti per laureati durano normalmente dai 2 ai 3 anni e permettono una formazione ad alto livello. I principali contratti di questo tipo sono la Fellowship, il Graduate Engineer Training ed il contratto Marie-Curie.

**STAFF** I contratti di Staff durano 5 anni e sono uno dei più ambiti e prestigiosi tipi di contratto al CERN come dipendente effettivo e professionista.

**E DOPO...** Purtroppo anche i contratti di Staff sono non rinnovabili ed a tempo determinato (5 anni). Tuttavia, verso gli ultimi anni di contratto, il CERN dà la possibilità al membro dello Staff di fare domanda per il cosiddetto Indefinite Contract (IC), ovvero un contratto di Staff a tempo indeterminato.

### 1.3.3 *La struttura generale*

Il consiglio del CERN è la più alta autorità dell'intera organizzazione ed è responsabile di tutte le decisioni più importanti. Si occupa di controllare tutte le attività scientifiche, tecniche ed amministrative del CERN e di approvare programmi, attività, budget e spese. Il consiglio è composto da 21 stati membri, ognuno dei quali fornisce due delegati come rappresentanti nel consiglio. Uno dei due delegati rappresenterà gli interessi amministrativi del proprio governo, mentre l'altro rappresenterà gli interessi scientifici nazionali. Nel consiglio, ogni stato membro ha a disposizione un solo voto e per la maggior parte delle decisioni si vota per maggioranza. Il consiglio è assistito, nelle sue decisioni, dalla Scientific Policy Committee e dalla Finance Committee.

La Scientific Policy Committee (Commissione della Politica Scientifica) valuta i pro e i contro delle attività proposte dai fisici e suggerisce strategie per il programma scientifico al CERN. I membri vengono eletti dalla commissione stessa e suggeriti dal consiglio in base ai meriti scientifici, senza distinzioni di nazionalità. La Finance Committee (Commissione della Finanza) è invece composta da rappresentanti delle amministrazioni nazionali e si occupa di gestire i contributi degli stati membri e budget e spese dell'organizzazione.

Il Director General (DG), eletto dal consiglio ogni 5 anni, si occupa di gestire il laboratorio CERN, rispondendo direttamente al consiglio. Il DG è assistito da un direttorato e gestisce il laboratorio attraverso una struttura di dipartimenti. Al momento del periodo di Technical Student del sottoscritto il DG era il fisico tedesco Rolf-Dieter Heuer, in carica dal 2009. Dal Gennaio 2016, invece sarà Fabiola Giannotti, fisica italiana e prima DG donna della storia.

Sotto al DG si sviluppa la struttura del laboratorio CERN, ovvero una struttura piramidale organizzata in dipartimenti. Più in particolare, il CERN è suddiviso in otto dipartimenti: BE (Beams), EN (Engineering), FP (Finance, Procurement and Knowledge Transfer), GS (General Infrastructure Services), HR (Human Resources), IT, PH (Physics), TE (Technology). Ogni dipartimento (gestito da un Department Head) è suddiviso in gruppi, che si occupano di settori specifici all'interno dello stesso dipartimento. A loro volta, ogni gruppo (gestito da un Group Leader) si suddivide in sezioni, ognuna della quale si specializza ulteriormente a seconda del lavoro svolto. Infine, ogni sezione (gestita da un Section Leader) è composta da una serie di progetti. Ad ogni progetto è associato un team il quale, sotto la supervisione di un Project Manager, si occupa di portare a termine il proprio progetto nel modo migliore possibile. [3]

**INDICO** Il progetto Indico, oggetto di questa tesi, è così collocato all'interno della struttura del CERN: IT - CIS - AVC - Indico. Indico fa quindi, innanzitutto, parte del dipartimento IT, ovvero il dipartimento dei servizi informatici al CERN (il cui Department Head è Frédéric Hemmer). Quindi, Indico fa parte del gruppo CIS (Collaboration & Information Services) che si occupa dei servizi di

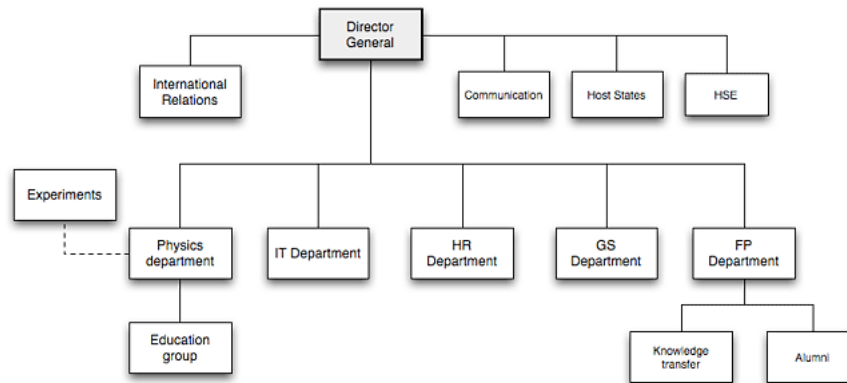


Figura 7: Organigramma ad alta granularità dell'organizzazione CERN.

collaborazione come videoconferenze, pubblicazioni elettroniche, e servizi di gestione dei documenti. Il Group Leader del gruppo CIS è Tim Smith. Più nel dettaglio, Indico fa parte della sezione AVC (AudioVisual and Collaborative Services), la quale si occupa di servizi per le videoconferenze e la gestione di eventi, sotto la supervisione del Section Leader Thomas Baron. Come vedremo più in dettaglio nel Capitolo 2, Indico è proprio il progetto che si occupa della gestione dell'applicazione web (o web application) per la creazione e gestione di eventi (come conferenze, meeting, ecc...). Durante la permanenza del sottoscritto al CERN, il Project Manager di Indico è stato prima Jose Benito Gonzales e quindi Pedro Ferreira, il quale è stato anche supervisore del sottoscritto per tutta la durata del progetto.

#### 1.3.4 Knowledge Transfer

Il gruppo Knowledge Transfer (KT), parte del dipartimento FP, si occupa di organizzare, catturare e distribuire la conoscenza e la tecnologia all'interno e fuori dal CERN. Uno degli obiettivi è quello, ad esempio, di applicare i risultati scientifici ottenuti al CERN in altri campi, come ad esempio la medicina (si stanno studiando, ad esempio, possibili tecnologie per la rimozione di tumori utilizzando piccoli acceleratori di particelle). Spesso, invece, quello che il gruppo KT si prefigge di fare è rendere qualche prodotto o tecnologia CERN più accessibile al mondo esterno, come università o aziende.

Proprio questo era il fine dei fondi che il gruppo KT ha destinato al progetto Indico. L'obiettivo era quello di rendere Indico più accessibile alle istituzioni e alle aziende che lo volessero utilizzare al di fuori del CERN e di renderlo più facile da utilizzare. Ma di questo parleremo ampiamente nella Parte II di questa tesi.

## INDICO

---

Indico è una web application che fornisce strumenti per l'organizzazione di eventi di qualsiasi dimensione, da semplici lezioni a grandi conferenze. Prese vita nel 2002 come progetto europeo, per poi venir adottato dal CERN nel 2004 come software per la gestione di eventi. Dal 2004 ad oggi, il CERN è stato il principale finanziatore del progetto Indico, dedicando un team al suo sviluppo e supporto.



Figura 8: Logo di Indico.

Indico è uno strumento open source<sup>2</sup>, il che vuol dire non soltanto che è completamente gratuito, ma anche che altre istituzioni o individui al di fuori del CERN possono ispezionarne il codice, contribuire allo sviluppo o modificarlo secondo le proprie preferenze.

Indico viene quindi utilizzato da centinaia di istituzioni e organizzazioni in tutto il mondo, ma il principale utilizzatore rimane il CERN stesso, che lo utilizza ogni giorno per gestire più di 300.000 eventi di diversa complessità e circa 200 stanze per conferenze e riunioni. L'istanza di Indico installata al CERN prende il nome di Indico@CERN, raggiungibile all'indirizzo <http://indico.cern.ch/>. Inoltre, al CERN è installata anche una seconda istanza minore di Indico utilizzata per prenotare gli uffici Burotel.

### 2.1 PRINCIPALI CARATTERISTICHE

Con Indico si possono gestire eventi di qualsiasi complessità: lezioni, riunioni, seminari e conferenze. Indico fornisce all'utente, durante tutta la fase di creazione e modifica di un evento, tutta una serie di strumenti per agevolare queste

---

<sup>2</sup> Il codice aggiornato di Indico, e di tutti i progetti ad esso correlati, può essere trovato all'indirizzo <https://github.com/indico>.

operazioni: l'utente può decidere secondo quali criteri far registrare terzi al suo evento, oppure come gestire l'invio di pubblicazioni o di materiale relativo all'evento.

**UI** Indico utilizza anche una potente e immediata interfaccia utente che si basa sul design WYSIWYG (What You See Is What You Get). Grazie a questa UI (User Interface) sarà facilissimo eseguire azioni altrimenti più complesse e tediose: la definizione di un form di registrazione per l'evento, la gestione delle timetable (Figura 9), che implementano un'intuitiva interfaccia drag-and-drop, oppure editor di testo che permettono l'utilizzo di rich text o formule matematiche.

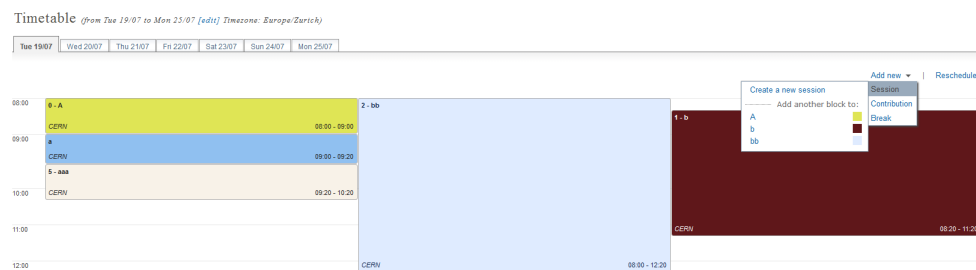


Figura 9: Un esempio di timetable di una conferenza in Indico.

**STRUTTURA** Indico, ed il suo sistema di protezione, è organizzato in una struttura ad albero per categorie. Indico è stato sviluppato principalmente per grandi aziende, per questo gli eventi, ed il materiale ad essi associato, sono organizzati secondo una struttura gerarchica di categorie. L'amministratore del sistema potrà decidere ed assegnare livelli di protezione alle varie categorie secondo diversi livelli di granularità.

**RICERCA** In Indico è incluso anche un potente sistema di ricerca, che permette di ricercare gli eventi desiderati tramite poche parole chiave o osservare gli eventi previsti per un determinato periodo di tempo. Inoltre, tramite la dashboard, è possibile accedere velocemente a tutti gli eventi a cui l'utente è iscritto o che sta osservando.

**ROOM BOOKING** Le compagnie e le organizzazioni, specialmente le più grandi, hanno spesso bisogno di gestire, limitare e tener traccia dell'utilizzo delle varie stanze e siti al loro interno. Per questo Indico include un potente ed intuitivo modulo per la prenotazione delle stanze (in inglese, room booking) che permette all'utente di specificare le caratteristiche di una stanza, approvare o meno la prenotazione di certe stanze e gestire strumenti e materiale messi a disposizione di certe stanze, come dispositivi audiovisivi per le video conferenze.



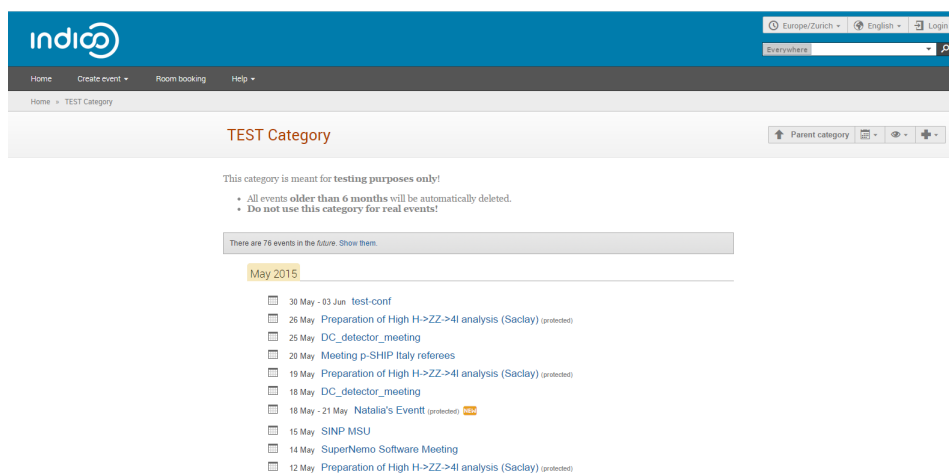


Figura 10: Un esempio di categoria con una serie di eventi al suo interno.

**CHAT & VIDEO** Per rendere la gestione online di riunioni e conferenze ancora più efficiente, Indico integra perfettamente Vidyo<sup>1</sup>, uno strumento per le videoconferenze, e permette di associare delle chatrooms Jabber/XMPP agli eventi.

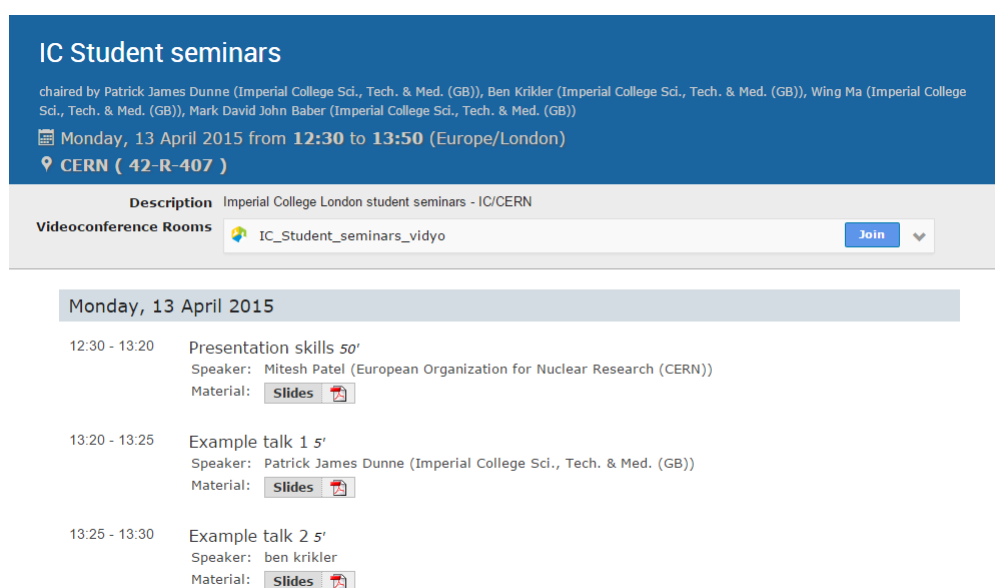


Figura 11: Un esempio di riunione in Indico con una videoconference room Vidyo associata.

<sup>1</sup> <http://it.vidyo.com/>.

**API** Dal momento che Indico è stato creato per agevolare la collaborazione e lo scambio di dati, tutte le informazioni contenute in Indico non sono esclusive di Indico stesso, ma possono essere recuperate (una volta accertato di avere l'accesso a quelle informazioni) tramite una semplice API (Application Programming Interface). Quest'API è garantita essere RESTful (dove REST significa Representational State Transfer) ed è in costante aggiornamento.

## 2.2 DETTAGLI TECNICI

Indico è un'applicazione web scritta principalmente in Python e Javascript (si veda la Figura 12).

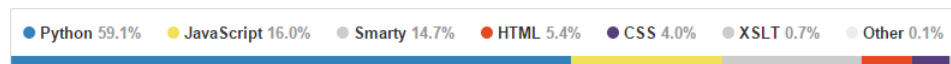


Figura 12: Una statistica dei linguaggi utilizzati nel codice di Indico (come mostrata su Github).

Essendo un'applicazione web, Indico implementa un web server ed un Database (DB) per gestire le richieste dell'utente e recuperare i dati richiesti.

**WSGI** Indico utilizza WSGI (Web Server Gateway Interface) per gestire il web server. WSGI è un'interfaccia standard tra web server e applicazioni Python, ovvero un'astrazione per rendere la comunicazione tra le due parti più semplice e modulare. In questo modo il server e l'applicazione sono completamente separati e seguono lo standard. Ad esempio per Indico@CERN si utilizza Apache come web server, ma un amministratore di un'altra istanza può scegliere un altro server se lo desidera, in quanto WSGI si configura senza problemi con quasi tutti gli web server in circolazione. [18]

**DB** Prima dell'inizio del periodo di Technical Student, Indico utilizzava ZODB (Zope Object Database) come database. ZODB è un database object-oriented che immagazzina oggetti Python. Dai primi del 2014 è invece stata iniziata una fase di migrazione, in quanto ZODB presentava alcuni svantaggi, come la difficoltà di eseguire query, la non indicizzazione dei record e il fatto di poter essere utilizzabile soltanto con Python. Dopo alcuni mesi di ricerca su quale potesse essere il miglior candidato per sostituire ZODB, si è scelto PostgreSQL (spesso detto Postgres) come vincitore. Postgres è un ORDBMS (Object-Relational Database Management System), ovvero un database relazionale a oggetti, che garantirà prestazioni più elevate durante l'utilizzo di Indico. Ovviamente passare da un database ad un altro non è una cosa semplice, in quanto tutto il codice dove si comunica con il DB dev'essere adeguato al nuovo database. Migrare l'intero DB di Indico@CERN in sol colpo era quindi un'impresa del tutto on fattibile. Si

è optato quindi per una migrazione modulare: il processo di migrazione durerà circa un anno, durante il quale Indico utilizzerà un sistema di gestione del DB ibrido ZODB+Postgres, migrando modulo per modulo. [16][6]

**FLASK** Flask è un micro framework per applicazioni web, scritto in Python e basato su Werkzeug e Jinja2. Uno dei principali vantaggi offerti da Flask è l'URL Routing, ovvero la possibilità di gestire gli URL in modo dinamico. Si possono associare diversi template per ogni tipo di richiesta ricevuta dal server, in modo da generare una pagina dinamica a seconda degli argomenti passati da Flask al template engine. [19]

**TEMPLATE ENGINE(S)** Un template engine è uno strumento che permette di creare documenti personalizzati e dinamici a partire da un template (ovvero un modello) di base e dei dati di input. In questo caso, quando si parla di template engines, ci si riferisce a strumenti che generano pagine web secondo un certo modello inserendovi dei dati dinamici. Indico ha utilizzato per molti anni Mako come template engine ma in questi ultimi anni sta lentamente passando a Jinja2. [17]

**ALTRE** A parte queste tecnologie citate, Indico ha iniziato a impiegare, nel corso degli ultimi anni, altri strumenti degni di nota, come ad esempio WTForms, per la gestione dei campi dei form (ad esempio form di registrazione), SQLAlchemy, ovvero un toolkit open source SQL, ed Alembic, uno strumento per la migrazione di database da utilizzare con SQLAlchemy.

## 2.3 LA COMMUNITY

Come già accennato, Indico è un software open source. Il suo codice sorgente più aggiornato è infatti disponibile su Github, all'indirizzo <https://github.com/indico/indico>. Il fatto di essere open source ha portato centinaia di organizzazioni e istituzioni, soprattutto nel campo della fisica delle particelle come ad esempio INFN (Istituto Nazionale di Fisica Nucleare) e IHEP (Institute of High Energy Physics), ad utilizzare Indico per la gestione dei propri eventi. Negli anni questo ha portato alla formazione di una vera e propria community di utilizzatori (user e admin) di Indico.

Questa community non soltanto utilizza Indico per i propri eventi, ma contribuisce anche allo sviluppo stesso del software, inviando segnalazioni agli sviluppatori del team Indico al CERN sotto forma di ticket. Un ticket indica un problema da risolvere, o qualcosa da migliorare; risolvere il problema in questione o apportare la miglioria richiesta viene indicato con il termine di "chiusura del ticket".

Altre volte invece, se chi si imbatte in un errore è uno sviluppatore a sua volta,

può capitare che provi a risolvere il problema da solo, lavorando sul codice di Indico. Una volta risolto, chi ha risolto il problema può inviare una cosiddetta *pull request* tramite Github (il repository ufficiale che ospita il codice di Indico) agli sviluppatori di Indico al CERN, in modo da indicare che quel problema è già stato risolto e possono includere (operazione di *merge*) le modifiche apportate al codice principale di Indico.

Parte II

INDICO KT PROJECT

## KT PROJECT: UNA PANORAMICA

---

In questo capitolo introdurremo i principali obiettivi del Progetto KT per Indico in modo da dare una panoramica generale dei sottoprogetti di cui il progetto principale è composto. Inoltre verranno esposti i principali strumenti e linguaggi utilizzati durante lo sviluppo del progetto.

### 3.1 PROGETTI PRINCIPALI

Il progetto finanziato dal gruppo KT per Indico, oggetto principale del programma di Technical Student a cui ha partecipato il sottoscritto, è nato come co-progetto tra i gruppi IT e KT. L'obiettivo principale di questo progetto è quello di migliorare la visibilità e l'impatto di Indico in tutto il mondo, in particolare al di fuori della comunità HEP (High Energy Physics), all'interno della quale Indico si è già ampiamente affermato.

Come si è già accennato nel Capitolo 1, il concetto di KT si basa sull'idea della diffusione e condivisione della conoscenza e degli strumenti necessari per ottenerla. L'idea alla base del progetto era quindi quella di modificare e migliorare il software Indico in modo da permettere una miglior diffusione dello stesso nel mondo. Per fare questo, il KT Project si era prefissato tre obiettivi generali da raggiungere:

- rendere Indico più accessibile agli utenti
- rendere Indico più semplice da utilizzare e personalizzare
- rendere Indico, in generale, più moderno e visivamente "attraente"

Gli obiettivi prefissati col Progetto KT erano quindi molto ampi e generali e, come ci può immaginare, anche piuttosto complessi da mettere in pratica. Per questa ragione il progetto che è stato assegnato al sottoscritto non era che il primo di una serie di progetti, finanziati dal gruppo KT, al fine di migliorare l'impatto a livello mondiale del software Indico. Infatti, al durante il periodo di 14 mesi passati a Ginevra dal sottoscritto, erano stati approvati e finanziati già due progetti dal gruppo KT per Indico: il primo, assegnato al sottoscritto, ed un secondo da assegnare ad un futuro membro del team Indico, molto probabilmente un altro Technical Student.

Il primo Progetto KT per Indico è stato quindi pianificato e suddiviso in una serie di sotto-progetti, i quali dovevano essere terminati durante i 12 (poi diventati 14) mesi del programma Technical Student (si veda [5] per la prima

stesura del progetto). Quattro di questi sotto-progetti sono risultati essere più importanti e complessi degli altri ed sono andati ad occupare gran parte del periodo di Technical Student. Di seguito ne parleremo brevemente per avere un'idea generale dei progetti principali del KT Project, mentre nei Capitoli successivi vedremo in dettaglio ognuno di essi.

#### 3.1.1 *Cloud Deployment*

La prima fase del Progetto KT riguardava il Cloud Deployment di Indico, ovvero l'automatizzazione del processo di installazione e configurazione di Indico sia in ambiente cloud che in ambiente virtuale.

Il progetto è durato circa un mese e mezzo, dal 23 Ottobre 2013 al 9 Dicembre 2013, ed ha portato alla scrittura di due script fabric, uno per la creazione di immagini virtuali ed uno di gestione remota, ed una recipe cloud-init, per il deployment su struttura cloud.

#### 3.1.2 *Distribuzione e Packaging*

#### 3.1.3 *Instance Tracker*

#### 3.1.4 *Conference Customization Prototype*

### 3.2 STRUMENTI E LINGUAGGI

Con questa ultima Sezione introduttiva, intendiamo fornire al lettore una serie di conoscenze e nozioni di base utili a capire il lavoro svolto con questo progetto. Indico infatti è composto da molti linguaggi diversi ed utilizza molti strumenti, sviluppati da terzi, senza i quali non potrebbe funzionare. È necessario quindi sapere quali sono e cosa fanno ognuno di questi strumenti, nonché essere a conoscenza dei linguaggi utilizzati.

#### 3.2.1 *Github*

#### 3.2.2 *Python*

Un importante comando offerto da python è `.format()` : questa funzione sostituisce a degli speciali *placeholder* (o segnaposti, in italiano), presenti nell'oggetto stringa sul quale viene eseguito, i valori associati ad ogni placeholder tramite un particolare dizionario, passato come unico argomento. I placeholder sono parole chiave, che identificano un parametro in modo univoco, racchiuse tra parentesi graffe. Il comando `.format()` funziona quindi come segue:

```
data = {'first': 'Hodor', 'last': 'Hodor!'}  
template = '{first} {last}'  
result = template.format(**data)
```

Il risultato salvato in `result` sarà quindi la stringa `'Hodor Hodor!'`.

### 3.2.3 *Cloud-init*

Cloud-init<sup>1</sup> è uno degli strumenti più utilizzati per l’inizializzazione e configurazione di server cloud. Tramite la compilazione di alcune semplici impostazioni, l’utente sarà in grado di avviare un nuovo server cloud specificando una serie di azioni da eseguire in automatico durante il primo avvio, come ad esempio eseguire determinati script, copiare alcuni file da remoto, installare pacchetti ed applicazioni necessarie, e così via. Cloud-init è installato di default su molte distribuzioni Linux, come Ubuntu, Fedora, Debian, CentOS, ecc. [11]

In poche parole, Cloud-init è un modulo che viene eseguito all’avvio di una macchina virtuale e permette di specificare delle azioni da eseguire tramite un file detto *user-data*. Un file di questo tipo, ovvero che permette di specificare una serie di azioni che verranno eseguite in automatico, viene detto *recipe* (ovvero “ricetta” in inglese). Infatti si parla di *cloud-init recipe* riferendosi ad una particolare configurazione da passare a cloud-init.

Per utilizzare una cloud-init recipe, è sufficiente specificare il file `user-data` generato quando si avvia il server sul cloud per la prima volta. Il comando da usare varia a seconda del Cloud Service Provider scelto. Per infrastrutture cloud basate su tecnologia OpenStack, ad esempio, è sufficiente eseguire il seguente comando da terminale:

```
$ nova boot --image ubuntu-cloudimage --flavor 1 --user-data user-data
```

Tramite il file `user-data` è possibile passare al modulo cloud-init una serie di file in diversi formati supportati, tra i quali:

- file compresso in formato `.gzip` ;
- file MIME (Multipurpose Internet Mail Extensions) multipart;
- bash script;
- file cloud-config.

<sup>1</sup> <https://launchpad.net/cloud-init>



In particolare, i file gzip possono essere utili per ridurre le dimensioni del file `user-data`, essendo questo limitato a 16KB. I file MIME multipart sono tipi di file composti che servono a raggruppare altri file, dei tipi sopra citati, in un unico file. I file di script servono ad eseguire una serie di comandi subito dopo il primo boot, mentre i file cloud-config sono particolari file utilizzati per copiare file in remoto sulla macchina sul cloud oppure per installare tutti i pacchetti aggiuntivi necessari.

Come vedremo nel Capitolo 4, le ricette cloud-init sono state molto utili per la fase di Cloud Deployment di Indico.

#### 3.2.4 *Fabric*

Fabric è una “libreria Python e uno strumento da linea di comando per rendere più semplice l’uso di SSH (Secure Shell) per applicazioni di deployment o operazioni di amministrazione di sistema”. [12]



Figura 13: Logo di Fabric.

Fabric fornisce una serie di comandi per l’esecuzione sia locale che remota di comandi shell (normali o tramite `sudo`), per l’upload o il download di file o per l’esecuzione di operazioni ausiliare, come richiedere un input all’utente o bloccare l’esecuzione di un’operazione in esecuzione.

Uno script fabric è quindi un semplice script python, dove però si possono utilizzare tutti i comandi messi a disposizione da Fabric. Due comandi fabric molto importanti sono `local(cmd)` e `run(cmd)`. Entrambi i comandi eseguono il comando `cmd` passato come argomento come comando shell, l’unica differenza tra i due è che `local(cmd)` esegue `cmd` in locale, ovvero sulla macchina che sta eseguendo lo script fabric, mentre `run(cmd)` esegue `cmd` su una macchina remota, opportunamente specificata. Inoltre un altro comando molto utile è il comando `put(file, dest)` che permette di copiare il file `file`, che si trova sulla macchina locale, nella destinazione `dest` all’interno della macchina remota, quindi mimando il comportamento del comando `scp` di bash.

Per permettere una stesura più semplice e meno ridondante di uno script fabric, è possibile specificare alcune opzioni in un particolare dizionario, detto *ambiente* e denotato dalla variabile `env`, in modo da non doverle ripetere ogni volta che si invoca un comando fabric. Ad esempio specificando un valore per il campo `env.hosts` possiamo definire una volta per tutte l'indirizzo di tutte le macchine remote su cui vogliamo eseguire i comandi passati a `run()`, senza dover stare a ripeterli inutilmente ogni volta.

Ogni script fabric deve specificare delle operazioni, dette *task*, che possono poi essere eseguite da linea di comando. Ogni task è una funzione python contrassegnata dal decoratore `@task`, che contrassegna le task (operazioni che possono essere eseguite dall'utente), da funzioni interne dello script.

Per poter essere eseguito, uno script fabric deve essere chiamato `fabfile.py`. Per eseguire quindi uno script fabric basta eseguire un comando della forma seguente:

```
$ fab task1 task2
```

Un comando come quello appena mostrato esegue prima la task `task1` e quindi `task2` definite nel file `fabfile.py`.

Fabric verrà utilizzato, all'interno del progetto di Cloud Deployment, per la stesura sia di uno script per la creazione automatica di immagini virtuali, che di uno script per la gestione remota di macchine cloud.

### 3.2.5 QEMU e KVM

QEMU (Quick EMUlator) è un emulatore e strumento di virtualizzazione open source. KVM (Kernel-based Virtual Machine) invece è anch'esso uno strumento di virtualizzazione e fornisce anche delle funzionalità di accelerazione hardware per sistemi Linux.

QEMU e KVM possono essere utilizzati in congiunzione, tant'è che KVM viene distribuito da anni assieme a QEMU. Si potrebbe pensare di utilizzare soltanto uno di questi due strumenti per lavorare con macchine virtuali, tuttavia, anche se QEMU fornisce un sistema di virtualizzazione completo e a se stante, per applicazioni pratiche è spesso necessario affiancargli KVM per migliorarne le performance. D'altro canto, KVM soltanto non fornisce tutte le funzionalità di un ambiente di virtualizzazione completo come QEMU.

QEMU e KVM sono stati utilizzati durante la fase di Cloud Deployment del progetto, ed in particolare nello script di creazione di immagini virtuali, come vedremo all'interno del Capitolo 4.

Per maggiori informazioni su QEMU e KVM si consultino [21] e [24].

3.2.6 *Virtualenv*

3.2.7 *Requests*

3.2.8 *Twitter Bootstrap*

3.2.9 *Flask*

TODO: parlare di `route()` e `render_template()`

3.2.10 *Celery*

3.2.11 *PostgreSQL e SQLAlchemy*

3.2.12 *Jinja2*

3.2.13 *jqPlot*

3.2.14 *jVectorMap*

## CLOUD DEPLOYMENT

---

I concetti di *cloud* e *virtualizzazione* stanno prendendo sempre più piede, nel corso degli ultimi anni, nel settore informatico ed in particolare nel campo di sviluppo software.

L'idea alla base del cloud si basa sul semplice fatto che molte volte un utente, o un amministratore di sistema, può voler essere in grado di eseguire una certa applicazione, senza però doversi preoccupare dell'hardware necessario. L'unica cosa che interessa è poter eseguire il software desiderato senza bisogno di doversi preoccupare dei dettagli tecnici quali installare un server, configurarlo, o anche pensare alle varie problematiche relative all'hardware come la capacità degli hard disk necessaria, di quanti processori ha bisogno, di quanta banda per la trasmissione dei dati, ecc. Per rispondere a questa esigenza, sono nati i *Cloud Service Provider*, ovvero aziende in possesso di un grandissimo numero di macchine, molto capienti, molto potenti e molto veloci. In parole povere, quello che offrono i Cloud Service Provider, è di affittare le loro macchine per un certo costo mensile (o annuale, dipende dal tipo di contratto). Questi provider fanno scegliere all'utente il tipo di profilo desiderato: ci saranno ad esempio i bassi profili, che forniscono risorse limitate in cambio di un pagamento minimo, oppure alti profili, che forniscono all'utente un'altissima potenza di calcolo in cambio, ovviamente, di un pagamento più alto. L'utente si limita quindi a scegliere il profilo che più gli è consono e a "lanciare" (in inglese, *deploy*) la propria applicazione sulla macchina appena affittata. Non sarà necessario quindi fare alcuna installazione fisica di macchine o hardware, né tanto meno mantenerle: il Cloud Service Provider scelto si occuperà di tutto questo, mentre l'utente potrà concentrarsi sulla gestione e sull'utilizzo della propria applicazione. Ovviamente, dato che facendo così l'utente si "astrae" dal concetto di macchina fisica, egli non saprà mai con certezza su che specifiche hardware viene effettivamente eseguita la sua applicazione. La sua applicazione potrebbe venire eseguita su un supercomputer molto potente, o magari su una macchina più semplice dedicata solo a quell'applicazione. L'utente non lo sa ma, d'altro canto, nemmeno gli interessa, avendo scelto di lanciare la sua applicazione sul cloud. Addirittura spesso può anche capitare che, a seconda del carico di lavoro delle varie macchine del provider, l'applicazione venga prima eseguita su delle macchine e in un secondo momento su delle altre. Da questo il cloud prende il suo nome: un'applicazione lanciata sul cloud non risiede necessariamente in una macchina specifica, ma la possiamo immaginare all'interno di una sorta di "nuvola", ovvero in uno spazio indefinito all'interno del quale però l'applicazione ha accesso a tutte le risorse hardware richieste.

L'idea della virtualizzazione è molto simile a quella del cloud ma orientata più al sistema operativo di una macchina. Sappiamo che spesso un'applicazione è ottimizzata per un certo sistema operativo o, addirittura, funziona soltanto se eseguita su determinati sistemi operativi. Per gli sviluppatori, spesso, è una scelta obbligata quella di prediligere alcuni sistemi operativi rispetto ad altri: infatti alcuni sistemi operativi possono essere talmente diversi tra loro che garantire la compatibilità dell'applicazione su tutti i sistemi operativi comporterebbe dover riprogettare e riscrivere l'applicazione da capo, il che non è sempre possibile, a seconda delle risorse disponibili per lo sviluppo. Per ovviare a questi problemi sono stati sviluppati degli appositi tools di virtualizzazione, come il ben noto Virtualbox di Oracle<sup>1</sup>. I tool di virtualizzazione permettono di simulare un sistema operativo eseguendolo su una macchina fisica sulla quale è installato un altro sistema operativo. Quindi è come se il tool di virtualizzazione simulasse un hardware che in realtà non c'è per dar modo all'utente di utilizzare un sistema operativo a sua scelta senza bisogno di doverlo installare sul disco fisso della macchina fisica. Questo permette di installare in modo molto veloce molti sistemi operativi diversi, qualora se ne avesse il bisogno. Ogni istanza creata tramite un tool di virtualizzazione prende il nome di *macchina virtuale*, detta anche Virtual Machine (VM) in inglese, e vengono spesso archiviate in appositi file (la cui estensione varia a seconda dello strumento di virtualizzazione scelto) che prendono il nome di *immagini virtuali*. Un altro vantaggio delle macchine virtuali è quindi la loro portabilità: se un utente volesse, infatti, utilizzare una certa macchina virtuale su un'altra macchina fisica, non dovrà far altro che copiarvi l'immagine virtuale relativa ed eseguirla sulla nuova macchina fisica tramite lo strumento di virtualizzazione.

Le tecniche di cloud e virtualizzazione sono quindi molto utili in quei frangenti in cui l'utente vuole solo occuparsi di poter eseguire la propria applicazione senza dover stare a preoccuparsi della configurazione hardware o del sistema operativo.

Con queste idee in mente, il primo obiettivo dell'Indico KT Project è stato proprio quello di poter adattare il software di Indico alle tecnologie di cloud e virtualizzazione. Può capitare, infatti, che un utente voglia installare ed utilizzare Indico per un breve periodo, senza stare a configurare un intero web server: a volte possono capitare utenti che vogliono utilizzare Indico per un solo evento o anche associazioni che intendono utilizzare Indico in modo continuativo ma che sono troppo piccole per occuparsi di installare e mantenere un server per conto proprio. Tutto questo era possibile già prima, ma se un utente aveva la necessità di installare Indico sul cloud doveva occuparsi personalmente di tutta la parte tediosa di installazione e configurazione, sia della macchina che di Indico, prima di poterne usufruire. Inoltre ogni interazione con la macchina sul cloud richiedeva l'utente di effettuare il login sulla macchina remota ed interagire

---

<sup>1</sup> <https://www.virtualbox.org/>

tramite terminale. Analogamente, se un utente avesse voluto installare Indico su una macchina virtuale, non avrebbe avuto altra scelta che farlo manualmente, dovendosi occupare personalmente dell'installazione e della configurazione sia della macchina virtuale che di Indico.

I principali obiettivi di questo progetto erano quindi di automatizzare il deployment su struttura cloud, da un lato, e la creazione di immagini virtuali, dall'altro. Successivamente è stato anche creato uno script in python fabric per la gestione remota (ad esempio sul cloud) di una macchina con installato Indico.

Il progetto, disponibile sulla pagina Github ufficiale di Indico<sup>1</sup>, è strutturato come segue:

- all'interno della cartella `usr/` sono presenti lo script per la generazione del file `user-data` e lo script fabric per la gestione remota dei server cloud;
- nella cartella `dev/` è presente lo script fabric per la generazione di immagini virtuali;
- la cartella `tpl/` raccoglie tutti i template necessari ai vari script;
- nella cartella `conf/` sono presenti alcuni file di configurazione.

Dal momento che al CERN Indico è installato su una macchina con sistema operativo Scientific Linux CERN 6 (SLC6)<sup>2</sup>, ovvero una distribuzione di Linux sviluppata al CERN basata su Scientific Linux 6 (SL6)<sup>3</sup>, si è deciso, per motivi pratici, di basare gli script di cloud deployment proprio su questa distribuzione di Linux. In linea teorica, gli script dovrebbero funzionare senza problemi anche per altre distribuzioni basate, come SLC6 e SL6, su Red Hat Enterprise Linux (RHEL). Per altre distribuzioni Linux potrebbero esser necessarie delle modifiche agli script, come ad esempio quando vengono invocati i comandi per l'installazione di pacchetti<sup>4</sup>.

#### 4.1 DEPLOYMENT CON CLOUD-INIT

Il primo obiettivo del progetto di cloud deployment di Indico era appunto quello di automatizzare l'installazione e la configurazione di un server sul cloud e di Indico. Per fare ciò abbiamo sfruttato le potenzialità di uno strumento di cloud configuration molto diffuso in ambiente Linux: il modulo cloud-init. Abbiamo già introdotto cloud-init in Sezione 3.2, a pagina 22, quindi eviteremo di ripetere

<sup>1</sup> <https://github.com/indico/indico-cloud-images>.

<sup>2</sup> <http://linux.web.cern.ch/linux/scientific6/>.

<sup>3</sup> Una distribuzione di Linux sviluppata dal Fermi National Accelerator Laboratory (FNAL), disponibile all'indirizzo: <https://www.scientificlinux.org/>.

<sup>4</sup> Il comando per sistemi RHEL è `yum` mentre per sistemi, ad esempio, Ubuntu Linux è necessario utilizzare il comando `apt-get`.

qui a cosa serve cloud-init e quali sono le sue funzionalità. Parleremo invece di come è stato utilizzato cloud-init per il cloud deployment automatizzato di Indico.

Le idee alla base dell'automatizzazione del cloud deployment di Indico possono essere riassunte dalle seguente necessità che un utente, in procinto di installare Indico su un nuovo server cloud, si trova a voler soddisfare:

- la necessità di configurare da zero, in poco tempo, un nuovo server cloud con Indico già installato e pronto all'uso;
- la necessità di poter ripetere questo processo in modo automatico;
- la necessità di parametrizzare alcune parti del processo di configurazione (sia del server che dell'applicazione);
- la necessità di fare tutto questo in modo sicuro e affidabile.

Come abbiamo già accennato in Sezione 3.2, la risposta a queste necessità è cloud-init.

Abbiamo già detto che cloud-init funziona passando un apposito file, o ricetta, detto `user-data`, al comando che si occupa di avviare il server cloud in remoto. Ovviamente il comando specifico varia a seconda del Cloud Service Provider scelto, ma solitamente i comandi dei provider che supportano cloud-init presentano un'opzione tramite la quale è possibile specificare il file `user-data` contenente tutte le informazioni ed istruzioni necessarie a configurare la nuova macchina cloud.

Il lavoro necessario, quindi, per automatizzare il tutto, si è risolto con lo scrivere uno script per la generazione del file `user-data`. Andiamo ad analizzare come è composto questo script ed il file `user-data` risultante.

Il file `user-data` necessario ai nostri fini è un file MIME multipart, ovvero un file in grado di raggruppare una serie di altri file. Il file MIME generato è così composto:

- uno script bash per eseguire le istruzioni richieste;
- un file cloud-config per copiare i file necessari.

Lo script python che genera il file `user-data`, denominato `gen-user-data.py`, è quindi suddiviso in quattro fasi principali: la fase di configurazione, le fasi di generazione dello script bash e del file cloud-config ed infine la fase di generazione del file `user-data` vero e proprio.

#### 4.1.1 Fase di configurazione

La fase di configurazione consiste semplicemente in una serie di domande mostrate su linea di comando, ognuna delle quali serve a far scegliere all'utente

tutti quei parametri necessari per personalizzare la propria installazione di Indico sul nuovo server cloud. Tra i parametri che l'utente può scegliere ci sono ad esempio i percorsi delle varie directory di installazione di Indico, o le porte e gli indirizzi ai quali il web server di Indico sarà raggiungibile, o ancora i certificati SSL (Secure Sockets Layer) da utilizzare.

L'utente potrà scegliere, ad ogni domanda, di utilizzare il valore di default suggerito, oppure di specificare un nuovo valore per quel parametro. Inoltre, potrà anche scegliere di generare un file di configurazione, contenente tutti i valori scelti, in modo da poter rieffettuare, in futuro, lo stesso processo di deployment utilizzando gli stessi valori senza bisogno di doverli reinserire una seconda volta.

Eseguendo lo script `gen-user-data.py` l'utente si troverà quindi a dover impostare i seguenti parametri (si noti che per ogni voce è presentato un valore di default, per il quale basta premere Enter):

```
$ python gen-user-data.py
Do you want to use a configuration file [y/N]?
Insert the Indico installation directory path [/opt/indico]:
Insert the Indico DB installation directory path [/opt/indico/db]:
Insert the Apache conf directory [/etc/httpd/conf]:
Insert the Apache confd directory [/etc/httpd/conf.d]:
Insert the SSL cert directory [/etc/ssl/certs]:
Insert the SSL private directory [/etc/ssl/private]:
Do you want to load a personal SSL certificate [y/N]?
Insert the http port [80]:
Insert the https port [443]:
Insert the hostname:
Insert the iptables path [/etc/sysconfig/iptables]:
Insert the Redis hostname [localhost]:
Insert the Redis port [6379]:
Insert the Redis password:
Do you want to use Postfix as mail server [Y/n]?
Insert the SMTP server port [25]:
Insert the SMTP login:
Insert the SMTP password:
Insert the YUM repositories directory [/etc/yum.repos.d]:
Insert the priority for the puia-unsupported repository [19]:
Do you want to generate a configuration file [Y/n]?
Specify the configuration file path [gen-user-data.conf]:
Choose a path for the MIME file [user-data]:
```

Al termine del processo, i valori vengono comunque salvati all'interno di un dizionario python, che sarà poi utilizzato, assieme ai vari template, per generare i file necessari.



#### 4.1.2 Generazione dello script bash

Come già accennato, la generazione dello script bash da includere nel file `user-data` si basa su un template, chiamato `user-data-script.sh`, e sui valori dei parametri scelti dall'utente. Questo è necessario in quanto alcune parti dello script sono parametrizzate e devono essere compilate in base alle scelte dell'utente.

Per ottenere lo script finale, basterà allora eseguire il comando python `.format()` su ogni linea del template passando come unico argomento il dizionario, creato nella fase precedente, dei parametri. In particolare il template dello script è composto da tutte le istruzioni dello script finale ma in corrispondenza di ogni parametro vi sarà invece un particolare placeholder che sta a indicare dove il comando `.format()` dovrà andare a sostituire i valori effettivi dei parametri.

Le principali azioni intraprese dallo script, una volta avviata la macchina per la prima volta, sono:

1. scaricare e installare tutti i pacchetti necessari ad Indico;
2. installare e configurare Indico;
3. installare i certificati SSL;
4. aprire le porte scelte per il web server;
5. copiare i file di configurazione nelle cartelle corrispondenti.

Avendo basato lo script su SL6, l'installazione di pacchetti aggiuntivi avviene invocando il comando `yum`. L'installazione e la configurazione di Indico avvengono tramite i comandi `easy_install indico` e `indico_initial_setup`, rispettivamente. La copia dei certificati SSL e dei file di configurazione e l'apertura delle porte, invece, avvengono tramite semplici comandi per la manipolazione di file in ambiente Linux.

I problemi principali riscontrati durante la stesura dello script riguardavano tutti l'impossibilità (apparente) di rendere automatiche alcune azioni. Per alcune parti dello script sono stati infatti necessari alcuni accorgimenti per rendere il procedimento pienamente automatico.

Per quanto riguarda l'installazione tramite comando `yum`, ad esempio, si è dovuta aggiungere l'opzione `-y` da linea di comando, per evitare che `yum` chiedesse conferma all'utente di voler effettivamente installare i pacchetti scelti, rimanendo ad aspettare all'infinito.

Un'altra problematica era legata al comando di configurazione di Indico, `indico_initial_setup`, che richiede all'utente di scegliere alcuni valori per configurare correttamente Indico. Nel nostro caso questi valori sono già stati scelti durante la fase di configurazione esposta prima, quindi devono essere passati in modo automatico al comando `indico_initial_setup`. La soluzione è far stampare

questi valori al terminale tramite il comando `echo` e quindi concatenare `echo` con `indico_initial_setup`. Il risultato è il comando seguente:

```
$ echo -e "{indico_inst_dir}\nc\ny\n{db_inst_dir}" | indico_initial_setup
```

Si notino i due placeholder `{indico_inst_dir}` e `{db_inst_dir}` che stanno a indicare, rispettivamente, il percorso in cui l'utente vuole installare Indico e in cui vuole installare il DB.

Infine è sorto il problema di dover far eseguire alcune istruzioni allo script come utente root, ovvero tramite il comando `sudo`. Il problema è che lo script non viene eseguito dall'utente, ma dal modulo cloud-init all'avvio della macchina sul cloud, senza possibilità di avere permessi da root. La soluzione è stata trovata utilizzando il comando `visudo` e andando a modificare il file `etc/sudoers` per permettere di eseguire `sudo` anche in assenza di TTY (Teletypewriter), ovvero una console. Il risultato<sup>1</sup> è il seguente frammento di codice che abilita il comando `sudo` anche all'interno di script cloud-init:

```
touch /etc/sudoers.tmp
cp /etc/sudoers /tmp/sudoers.new
find_replace /tmp/sudoers.new "Defaults requiretty" "Defaults !requiretty"
visudo -c -f /tmp/sudoers.new
if [ "$?" -eq "0" ]; then
    cp /tmp/sudoers.new /etc/sudoers
fi
rm /etc/sudoers.tmp
```

Ricapitolando, dopo aver generato lo script effettivo sostituendo i parametri ai rispettivi placeholder nel template, e con le dovute accortezze per rendere il tutto completamente automatico, il risultato è uno script che, durante il primo avvio della macchina sul cloud, si occuperà di effettuare tutte le azioni necessarie ad installare e configurare Indico, senza che l'utente debba fare niente. L'unica cosa di cui ha bisogno lo script è che i vari file di configurazione necessari ad Indico siano già stati copiati sulla macchina: a questo penserà il file cloud-config, generato nella prossima fase.

#### 4.1.3 Generazione del file cloud-config

Dopo aver generato lo script bash da inserire nel file MIME multiparti finale, lo script `gen-user-data.py` si occupa di generare il file cloud-config, necessario a copiare file sulla macchina cloud che vogliamo avviare. Come per lo script

<sup>1</sup> Si vedano <http://serverfault.com/questions/324415/running-sudo-commands-in-cloud-init-script> e <http://stackoverflow.com/questions/323957/how-do-i-edit-etc-sudoers-from-a-script>.

generato nella fase precedente, anche la generazione del file cloud-config si basa su dei template e sull'uso del comando `.format()`.

Il template del file cloud-config ha la seguente struttura:

```
#cloud-config

write_files:
- content: |
  {puias_repo_content}
  path: /puias.repo
- content: |
  {indico_httpd_conf_content}
  path: /indico_httpd.conf
- content: |
  {indico_indico_conf_content}
  path: /indico_indico.conf
- content: |
  {redis_conf_content}
  path: /redis.conf
- content: |
  {ssl_conf_content}
  path: /ssl.conf
{ssl_files}
```

Si vede quindi che per ogni file che dev'essere copiato è presente, nel template, una riga `- content: |`, che serve a delimitare l'inizio di un nuovo file. A seguito è presente un placeholder, che verrà sostituito con il contenuto del file in questione, ed infine la voce `path:`, nella quale si indica in che percorso della macchina cloud vorremmo copiare il file. La struttura del file cloud-config è riconosciuta tramite l'indentazione: la stringa `- content: |` non dev'essere indentata, il contenuto del file dev'essere indentato di 8 spazi mentre il percorso finale del file dev'essere indentato di 4 spazi.

Per comodità, tutti i file vengono inizialmente copiati nella cartella root `/` del sistema: ci penserà poi lo script bash generato alla fase precedente a copiare i file nelle rispettive cartelle lavorando direttamente sulla macchina cloud.

Dal momento che alcuni dei parametri scelti durante la prima fase potrebbero essere necessari anche all'interno dei file da copiare, sono stati scritti dei template per ognuno di questi file. Prima di generare il file cloud-config vero e proprio è quindi necessario generare i vari file che si vogliono copiare, tramite i corrispettivi template ed il comando `.format()`, come visto prima.

I file da copiare sono file di configurazione, come il file di configurazione di Indico, quello per la configurazione di Apache, o i certificati SSL.

Una volta generati tutti i file necessari a partire dai template e dai parametri scelti nella prima fase, lo script leggerà il contenuto di ogni file e lo scriverà, con

la dovuta indentazione, al posto del placeholder corrispondente nel template del file cloud-config. Il risultato finale sarà quindi il file cloud-config, riempito con il contenuto dei vari file che vogliamo copiare i quali, a loro volta, sono stati compilati con i parametri scelti dall'utente.

#### 4.1.4 Generazione del file `user-data`

Quando finalmente sono stati generati sia lo script bash che il file cloud-init, lo script principale può procedere all'ultima fase, ovvero mettere i due file insieme scrivendoli in un file MIME multipart, che rappresenterà il file `user-data` finale.

Per generare `user-data`, creando un nuovo file MIME multipart a partire dai due file generati nelle fasi precedenti, si è usato uno script reso disponibile dagli sviluppatori di cloud-init<sup>1</sup>, detto `write-mime-multipart`.

Questo comando, molto semplice, prende come input tutti i file che vogliamo includere nel file finale e richiede anche di specificare il nome e percorso del file che vogliamo venga generato. Quindi, nel nostro caso, il comando finale sarà il seguente:

```
$ ./write-mime-multipart --output user-data user-data-script.sh cloud-config
```

Chiaramente, lo script principale per la generazione del file `user-data` si occuperà di invocare il comando precedente, assicurandosi di passargli i valori corretti per i vari input. Dopodiché lo script terminerà.

Una volta che lo script avrà terminato con successo, l'utente potrà utilizzare il file `user-data` generato per inizializzare e configurare una nuova macchina su cloud in modo completamente automatico, semplicemente specificando il file `user-data` come argomento del comando di boot. Come già accennato in Sezione 3.2, il comando di boot effettivo, ed il metodo in cui si passa il file `user-data`, cambia a seconda del Cloud Service Provider scelto.

## 4.2 CREAZIONE DI IMMAGINI VIRTUALI

Il secondo obiettivo del progetto di Cloud Deployment è stato scrivere uno script per la creazione automatica di immagini virtuali già configurate per poter eseguire Indico.

Lo script in questione è uno script python fabric che si occuperà di avviare una nuova macchina virtuale, usando due strumenti di virtualizzazione (QEMU e KVM), e di effettuare le necessarie operazioni all'interno di essa. Tutte i parametri che l'utente può scegliere, come nome e percorso dell'immagine di base o le

<sup>1</sup> Reperibile all'indirizzo <https://github.com/lovelysystems/cloud-init/blob/master/tools/write-mime-multipart>

porte da aprire sulla macchina virtuale, sono raccolti all'interno di un file esterno allo script, chiamato `fabfile.conf`, in modo che l'utente li possa cambiare in modo semplice e veloce.

Le task specificate da questo script `fabfile` sono le seguenti:

- `create_vm_image` : crea un'immagine virtuale con Indico installato e configurato;
- `run_vm_debug` : avvia la macchina virtuale e Indico (in modalità debug);
- `config_no_cloud` : configura il file di configurazioni "fasullo" no-cloud;
- `launch_vm` : avvia la macchina virtuale;
- `start` : avvia Indico;
- `deploy` : esegue tutte le installazioni e configurazioni necessarie per far funzionare Indico sulla macchina virtuale;
- `config` : configura Indico e la macchina virtuale;
- `vm_config` : configura la macchina virtuale;
- `indico_config` : configura Indico;
- `indico_inst` : installazione e setup di indico;
- `dependencies_inst` : installazione delle dipendenze.

La task principale è ovviamente `create_vm_image`, che restituisce un'immagine virtuale con indico Installato e funzionante pronta ad essere avviata o caricata su una struttura cloud. Tutte le altre task sono sotto-task di `create_vm_image` che, all'occorrenza, possono essere invocate in modo autonomo.

`create_vm_image` è divisa in due fasi: nella prima fase si avvia una nuova macchina virtuale a partire da un'immagine virtuale di base; nella seconda si fanno tutte le installazioni e configurazioni necessarie per poter usare Indico sulla macchina virtuale.

#### 4.2.1 *Avvio della macchina virtuale*

La fase di avvio della macchina virtuale si divide, a sua volta, in due parti: configurazione di un'immagine `.iso` no-cloud e boot della macchina virtuale.

La fase no-cloud serve creare un'immagine `.iso` di configurazione per poter avviare la macchina virtuale anche al di fuori da una struttura cloud. Questa fase è necessaria in quanto il sistema operativo SLC6, che è stato utilizzato per sviluppare e testare lo script di creazione di immagini virtuali, è un sistema

operativo pensato per operare in ambiente cloud e non su ambienti di virtualizzazione locali. Nel nostro caso, invece, l'idea è di avviare una nuova macchina virtuale in locale (usando uno strumento di virtualizzazione) e di effettuare tutte le operazioni necessarie tramite uno script fabric. Quindi è necessario, quando si avvia per la prima volta questa macchina virtuale, "ingannarla" in qualche modo, facendole ignorare il fatto che non viene eseguita in ambiente cloud. Per fare questo utilizziamo la funzionalità no-cloud offerta dal modulo cloud-init<sup>1</sup> che abbiamo descritto nella Sezione precedente. Senza scendere nel dettaglio, questa fase genera due file, `user-data` e `meta-data`, e li archivia all'interno di un'immagine in formato `.iso`. Quest'immagine, che chiameremo `init.iso`, verrà passata, nella fase successiva, al comando di boot della macchina virtuale per poterla avviare senza problemi anche in ambienti di virtualizzazione locali.

Una volta creato il file `init.iso` per attivare la funzionalità no-cloud, lo script potrà finalmente avviare la macchina virtuale. Il comando per l'avvio è il seguente:

```
$ kvm -m 256 --redir tcp:2222::22 --net nic --net user, --drive file=slc6_cern_x86_64.qcow2,if=virtio --drive file=init.iso,if=virtio --serial file:qemu-output.log &
```

Con l'opzione `--drive file=` si specificano le immagini virtuali da usare come input: in questo caso utilizziamo `slc6_cern_x86_64.qcow2`, che è l'immagine virtuale di base con installato SLC6, e `init.iso`, che permette di attivare il no-cloud. L'opzione `--redir` serve ad associare le porte specificate, in modo da permettere la comunicazione tra la macchina locale e la macchina virtuale tramite quelle porte (nello specifico quelle sono le porte relative al protocollo SSH). Infine, l'opzione `--serial file:` permette di specificare un file di log, dove verrà salvato l'output del terminale della macchina virtuale, mentre il simbolo `&` finale serve a lanciare il comando in background, in modo da lasciare la shell libera e non bloccare l'esecuzione dello script.

Il file di log passato al comando `kvm` è molto importante, in quanto consente allo script fabric di sapere quando la macchina virtuale ha terminato la fase di booting, in modo da poter proseguire ed effettuare le azioni sulla macchina necessarie. Lo script fabric, infatti, non può eseguire azioni come installare pacchetti aggiuntivi o installare Indico finché la macchina si sta ancora avviando. D'altro canto, la macchina virtuale non ha alcun modo diretto per comunicare allo script fabric quando la fase di booting è finita. L'idea è stata quindi quella di far loggare l'output del terminale della macchina virtuale sul file di log specificato e far leggere il file allo script fabric in continuazione finché non viene rilevata la fine del boot, nel qual caso lo script può continuare la sua esecuzione. Nello specifico, per capire quando la fase di boot è terminata o meno, è bastato fare un boot di prova e andare a vedere qual era l'ultima riga scritta sul file

<sup>1</sup> Si veda <http://cloudinit.readthedocs.org/en/latest/topics/datasources.html#no-cloud>.

di log durante la fase di booting e quindi far controllare allo script `fabric` se tale stringa è presente o meno nel file di log: se la stringa compare nel log, allora lo script può procedere con l'esecuzione, altrimenti aspetta 5 secondi e poi ricontrolla. Il comando relativo a questa procedura è il seguente:

```
$ while ! grep -q "Starting atd:.*[*OK.*]" "qemu-output.log"; do sleep 5; done
```

Prima di passare alla fase successiva, osserviamo che l'immagine di base, nell'esempio `slc6_cern_x86_64.qcow2`, deve essere ottenuta dall'utente in maniera autonoma, ad esempio scaricandola tramite il sito ufficiale del sistema operativo scelto.

#### 4.2.2 Configurazione della macchina virtuale

Durante la fase di configurazione della macchina virtuale lo script esegue tutte quelle istruzioni necessarie ad installare pacchetti aggiuntivi e ad installare e configurare Indico e la macchina virtuale affinché Indico sia pienamente utilizzabile al termine.

Le istruzioni eseguite in questa fase dallo script `fabric` sono quindi del tutto analoghe e speculari a quelle eseguite dallo script `bash` che avevamo descritto nella Sezione 4.1 per il deployment di Indico su cloud tramite `cloud-init`. Non staremo quindi a ripetere le azioni intraprese dallo script in questa fase, essendo del tutto uguali a quelle dello script `bash` già descritto.

Le uniche differenze riguardano il fatto che, in questo caso, le istruzioni sono eseguite tramite uno script `fabric`. Quindi tutte le istruzioni eseguite dallo script `bash` in modo diretto dovranno adesso essere eseguite dallo script `fabric` in remoto, ovvero tramite il comando `run()`. Inoltre, non sarà più necessario, come prima, aver bisogno di copiare i file sulla macchina virtuale tramite meccanismi come `cloud-config` (sempre visto nella Sezione precedente): in questo caso, grazie a `Fabric`, sarà sufficiente eseguire il comando `put()` che si occuperà di copiare i file richiesti al momento in cui lo script lo richiede.

Riguardo al comando `put()` è stata tuttavia necessaria una modifica in quanto esso non supporta, al momento, di passare un link simbolico come percorso di destinazione del file. Per aggiungere il supporto a link simbolici, abbiamo inserito la seguente variante del comando, da utilizzare al posto di `put()`:

```
def _putl(source_file, dest_dir):
    """
    To be used instead of put, since it doesn't support symbolic links
    """

    put(source_file, '/')
    run("mkdir -p {0}".format(dest_dir))
```

```
run("mv -f /{0} {1}".format(os.path.basename(source_file), dest_dir))
```

Al termine dell'esecuzione dello script, l'immagine virtuale di base risulterà aggiornata con l'installazione e la configurazione di Indico (e di tutte le componenti necessarie ad Indico) e sarà pronta per essere utilizzata in locale (tramite strumenti di virtualizzazione) o su una struttura cloud.

#### 4.3 SCRIPT DI GESTIONE REMOTA

L'ultima parte di questo progetto riguarda lo script di gestione remota di macchine cloud per l'esecuzione di Indico. Questo script non era previsto tra gli obiettivi iniziali del progetto, ma è stato scritto, per motivi pratici, durante la stesura degli altri due script e alla fine, data la sua utilità, si è deciso di includerlo nella versione finale del codice del progetto di Cloud Deployment.

Lo script di gestione remota è uno script fabric che definisce delle task per la gestione e la configurazione in remoto di Indico e di tutte le componenti ad esso connesse. Può capitare, ad esempio, che l'utente decida, in un certo momento, di voler cambiare l'indirizzo del web server su cui gira Indico, o le porte tramite le quali si accede ai vari servizi, o magari vuol semplicemente arrestare Indico e farlo ripartire in un secondo momento. Tutto questo sarebbe già possibile, ma sarebbe necessario che l'utente acceda alla macchina remota, ad esempio tramite SSH, e dal terminale esegua i vari comandi manualmente. Questa procedura può essere tediosa e complicata e potrebbe anche dar luogo a errori. Per questo motivo è stato scritto questo script di gestione remota che definisce una serie di task, che l'utente può eseguire, che rappresentano i comandi principali che l'utente può voler eseguire sulla macchina remota e su Indico.

Di seguito elenchiamo le task implementate, senza scendere ulteriormente in dettaglio:

- `restart` : permette di riavviare una o più componenti di Indico (valori accettati sono `redis`, `db`, `httpd` e `postfix`);
- `start` : avvia una o più componenti di Indico (accetta gli stessi valori di `restart`);
- `config` : configura la macchina virtuale con tutte le informazioni necessarie;
- `update_smtp` : aggiorna la configurazione di Indico per SMTP (Simple Mail Transfer Protocol);
- `update_redis` : aggiorna la configurazione di Indico riguardante Redis;
- `update_server` : aggiorna le configurazioni del web server (come hostname e porte);



- `load_ssl` : carica dei nuovi certificati SSL.

## DISTRIBUZIONE E PACKAGING

---

Il software Indico, come già accennato, è un software open source, principalmente sviluppato al CERN. Il codice di Indico viene aggiornato regolarmente, producendo ogni volta nuove versioni di Indico. Data la vastità della community di Indico, ogni volta che viene rilasciata una nuova versione il team di Indico al CERN deve occuparsi di creare una distribuzione di Indico relativa a quella versione e quindi di distribuirla attraverso i principali canali di comunicazione. Non solo, un amministratore di una qualche istanza Indico, o magari uno sviluppatore esterno al gruppo Indico al CERN, potrebbe sentire il bisogno di creare una distribuzione di una specifica versione di Indico per testarla o per caricarla dove più desidera. Inoltre uno sviluppatore potrebbe avere bisogno di creare diverse distribuzioni per diverse versioni di Python o anche distinguere tra distribuzione del codice sorgente e distribuzione binaria precompilata, pronto ad essere eseguita.

Creare una nuova distribuzione ogni volta, personalizzandola alle proprie esigenze, e distribuirla su uno o più canali, risulta essere un processo tedioso e ripetitivo, specialmente se, per motivi di testing, si è costretti a generare molte distribuzioni in rapida successione.

Con questa seconda fase del Progetto KT per Indico si è cercato proprio di risolvere questo problema, cercando di automatizzare e parametrizzare il processo di generazione e diffusione di una distribuzione Indico.

L'obiettivo di questo progetto era quindi quello di scrivere uno script che automatizzasse:

- la creazione di distribuzioni di diverse versioni di Indico,
  - sia del codice sorgente (tarball),
  - che precompilata (Python egg);
- la creazione di distribuzioni binarie per diverse versioni di Python;
- l'upload della distribuzione
  - su un server
  - e su Github.

Con *tarball* si indica comunemente la distribuzione di un pacchetto o libreria i cui file sono archiviati tramite il comando `tar`, il quale genera un file in formato `.tar`. Spesso, come in questo caso, i file `.tar` vengono anche compressi utilizzando il comando `gzip`, producendo un file in formato `.tar.gz`. Solitamente

le tarball vengono utilizzate per distribuire codice sorgente di programmi open source, come in questo caso. I *Python egg* rappresentano invece distribuzioni di programmi Python precompilati. I Python egg sono molto utili quando non interessa il codice sorgente, in quanto per installarli basta eseguire il comando `easy_install` messo a disposizione da Python stesso. Per maggiori informazioni su tarball e Python egg si vedano [26] e [32].

Lo script scritto per automatizzare la creazione di una distribuzione e l'upload è uno script Fabric (si veda la Sezione 3.2), denominato `fabfile.py`, che è stato incluso nel codice sorgente del progetto principale di Indico<sup>1</sup>. Questo script era già presente prima di questo progetto ed era utilizzato per generare distribuzioni di codice sorgente e binario e per installare le dipendenze esterne necessarie a Indico. Si sono quindi aggiunte allo script le funzioni per generare distribuzioni per diverse versioni di Indico e di Python e per caricarle su un server e su un repository Github.<sup>2</sup>

La task principale di questo script Fabric si chiama `package_release` e la sua esecuzione si divide in due fasi: nella prima si genera la distribuzione secondo i parametri richiesti, mentre nella seconda si effettua l'upload della distribuzione generata su un server specificato e/o su un repository Github. Vediamo, nelle Sezioni seguenti, come funzionano queste due fasi.

## 5.1 GENERAZIONE DI UNA DISTRIBUZIONE

Durante la prima fase dello script di packaging, viene effettuata la generazione della distribuzione. Ad ogni esecuzione lo script permette di specificare la versione di Indico rispetto alla quale vogliamo creare la distribuzione ed una o più versioni di Python rispetto alle quali vogliamo creare le distribuzioni binarie. Prima di questo progetto, lo script era già in grado di generare tarball e Python egg, ma era limitato a farlo per la versione corrente di Indico e la versione di default di Python.

Lo script, innanzitutto, si preoccupa di selezionare i file sorgenti relativi alla versione Indico selezionata. Per far questo effettua un `git clone`, nel caso in cui il repository non sia presente in locale, e quindi effettua un `git checkout` in modo da cambiare branch e posizionarsi nella versione desiderata. Il sistema di nomenclatura dei branch in Indico segue la convenzione che ad ogni branch corrisponde una versione ed il nome del branch è formato da una "v" seguita dal numero della versione (eventualmente con punti per distinguere tra versioni principali e versioni minori). Il frammento di codice relativo a questa parte è il seguente:

<sup>1</sup> <https://github.com/indico/indico>.

<sup>2</sup> Per vedere in dettaglio le aggiunte apportate allo script da questo progetto di veda il seguente commit su Github: <https://github.com/indico/indico/commit/75ea63df8d19402d8ab47b22d77d1f7b4bf1fc79>.

```

with lcd(build_dir):
    if os.path.exists(os.path.join(build_dir, 'indico')):
        print yellow("Repository seems to already exist.")
        with lcd('indico'):
            local('git fetch {}'.format(upstream))
            local('git reset --hard FETCH_HEAD')
            if not no_clean:
                local('git clean -df')
    else:
        local('git clone {}'.format(upstream))
    with lcd('indico'):
        print green("Checking out branch \'{0}\''".format(indico_version))
        local('git checkout {}'.format(indico_version))
        local('fab _package_release:{},{},{}'.format(
            os.path.join(build_dir, 'indico'),
            ':'.join(py_versions), str(system_node).
            lower()))

```

Si vede che una volta effettuato il checkout del branch corretto (o `master` se non viene indicato niente) lo script invoca la funzione privata `_package_release`, che si occupa di generare la distribuzione dei sorgenti (tarball) e la/e distribuzione/i binaria/e per ogni versione Python selezionata (sotto forma di Python egg). Le funzioni per la generazione della tarball e del Python egg erano già presenti nello script e sono state riutilizzate, aggiungendo soltanto la facoltà di generare Python egg per diverse versioni di Python.

Per generare la distribuzione dei sorgenti, lo script installa e configura le dipendenze esterne di Indico e quindi procede con l'invocazione del comando `sdist` del modulo `setuptools` [23], che si occupa di generare una distribuzione dei sorgenti (il formato di default è `.tar.gz`, ovvero una tarball).

La generazione delle distribuzioni binarie è molto simile, con l'unica differenza che viene generata una distribuzione per ogni versione Python selezionata. In particolare, lo script utilizzerà di volta in volta un compilatore Python differente, a seconda della versione scelta, e, per ognuno di essi, invocherà il comando `bdist_egg`, del modulo `setuptools` [22], che genera una nuova distribuzione binaria sotto forma di Python egg. Per utilizzare compilatori Python differenti a seconda della versione desiderata, all'inizio dello script vengono creati tanti ambienti Python virtuali, tramite `virtualenv`, quante sono le versioni Python richieste: in ognuno di questi ambienti verrà installata una diversa versione di Python e, al momento della generazione del Python egg, non verrà semplicemente chiamato il compilatore Python globale (come veniva fatto nello script vecchio) ma si invocherà il compilatore specifico alla versione passata come argomento.

## 5.2 UPLOAD DELLA DISTRIBUZIONE

Una volta generate le distribuzioni desiderate, l'utente può scegliere se caricarle su Github, su un server oppure su entrambi. Tutte le impostazioni relative a questa fase possono essere impostate tramite il file di configurazione dello script, chiamato `fabfile.conf`.

### 5.2.1 Upload su Github

Github rappresenta il principale mezzo di distribuzione di Indico ed è uno strumento molto potente in quanto fornisce agli sviluppatori molti strumenti di distribuzione. In Github si possono infatti creare delle *release*, ovvero una distribuzione del software relativa ad una versione particolare. Per ogni release, inoltre, gli sviluppatori possono caricare allegare alcuni file, detti *asset*, caricandoli su Github, in modo che gli utenti possano usufruirne. L'obiettivo della fase di upload a Github prevede infatti di caricare le distribuzioni generate nella fase precedente come asset della release corrispondente alla versione di Python generata.

I parametri relativi a questa fase all'interno del file `fabfile.conf` hanno la seguente struttura:

```
github = {
    'user': 'your-username',
    'org': 'indico',
    'repo': 'indico',
    'auth_token': 'your-access-token',

    # What do to in case release exists
    # True: overwrite, False: do not overwrite, None: ask
    'overwrite': None
}
github['upstream'] = "https://github.com/{0}/{1}.git".format(github['org'], github['repo'])
```

La fase di upload su repository Github prevede innanzitutto che l'utente specifichi un username e password validi per poter accedere al repository richiesto. Alternativamente alla password, l'utente può anche specificare un token OAuth valido.

Una volta assicuratosi che le credenziali sono valide, lo script effettua delle richieste HyperText Markup Language (HTML), tramite il pacchetto `requests`, per controllare se la release corrispondente alla versione Indico selezionata è già presente o meno sul repository. Nel caso in cui sia già presente, e l'utente abbia settato a `True` il parametro `overwrite`, allora si procede a sovrascrivere gli asset

della release già esistente con le nuove distribuzioni generate. In caso contrario, si crea una nuova release e si caricano tutte le distribuzioni direttamente. Per fare l'upload di ogni singolo file generato si deve inviare una richiesta, contenente il file come dati, alla seguente API messa a disposizione da Github:

```
https://uploads.github.com/repos/{0}/{1}/releases/{2}/assets
```

Per una documentazione più dettagliata delle API di Github si veda [13].

A titolo d'esempio si può vedere la release 1.2 di Indico, raggiungibile all'indirizzo <https://github.com/indico/indico/releases/tag/v1.2.0>, nella quale sono state caricate automaticamente una distribuzione del codice sorgente ("Source code (tar.gz)") e due distribuzioni binarie, relative a Python 2.6 e 2.7 ("indico-1.2-py2.6.egg" e "indico-1.2-py2.7.egg"), utilizzando lo script appena descritto. Il riepilogo degli asset disponibili per la versione 1.2 viene visualizzato su Github come in Figura 14.

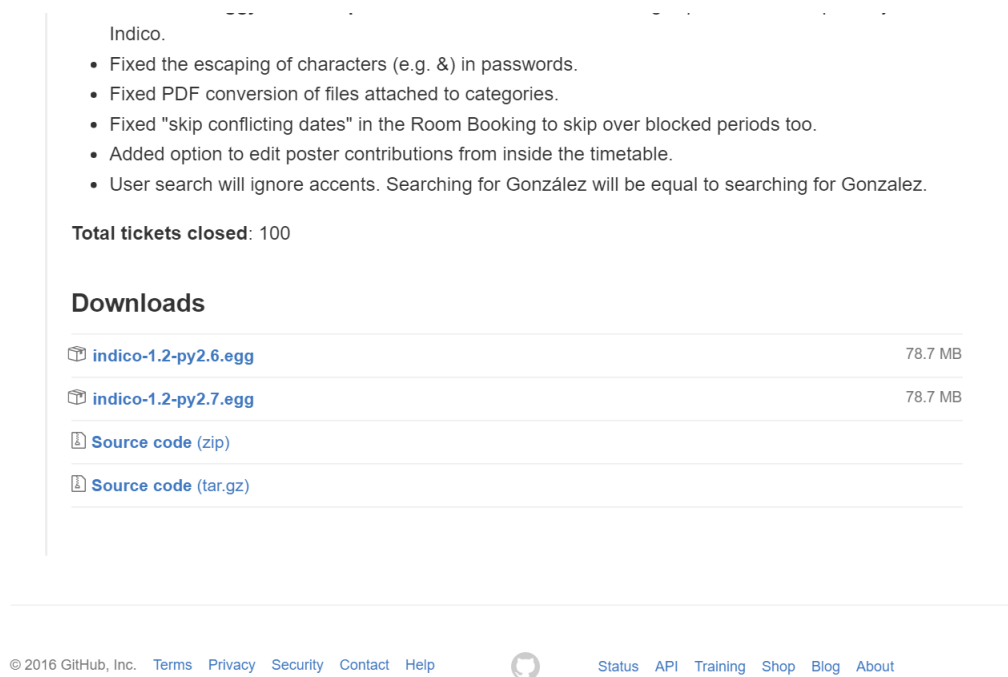


Figura 14: Asset caricati automaticamente per la release 1.2 di Indico su Github.

### 5.2.2 Upload su un server

L'upload a server è stato incluso come mezzo di diffusione alternativo di una distribuzione. Esso risulta molto più semplice rispetto all'upload su Github in

quanto richiede soltanto di specificare l'indirizzo e la porta di accesso del server e i dati di autenticazione (username e chiave SSH). Lo script si limiterà quindi ad invocare il comando `put()` di Fabric per copiare le distribuzioni generate nel percorso specificato sul server remoto.

Le configurazioni per l'upload a server devono seguire la seguente struttura all'interno di `fabfile.conf`:

```
ssh = {  
    'host': 'remote-server.example.com',  
    'port': '22',  
    'user': 'root',  
    'key': '~/.ssh/id_rsa',  
    'dest_dir': '/path/on/remote/server'  
}
```

## INSTANCE TRACKER

---

Indico è un software open source sviluppato al CERN. Come già accennato, Indico non è soltanto sviluppato al CERN ma viene anche utilizzato al CERN. L'istanza di Indico che gestisce gli eventi al CERN prende il nome di Indico@CERN. Inoltre, sempre al CERN, è presente un'altra installazione di Indico, denominata Burotel.

Ma Indico non è installato ed utilizzato esclusivamente al CERN: infatti molte altre istituzioni e organizzazioni di tutto il mondo ne fanno uso, come ad esempio INFN, Fermilab, IHEP o ILC (International Linear Collider). Alcune statistiche raccolte negli anni passati mostrano che Indico è installato in oltre 100 istituzioni e organizzazioni in tutto il mondo, specialmente nel settore della fisica delle particelle.

Come già accennato, il team di sviluppatori di Indico al CERN non si occupa soltanto di sviluppare e aggiornare il codice di Indico, ma anche di fornire assistenza a tutti gli utenti ed amministratori di altre istanze di Indico. Con una community così vasta, risulta tuttavia difficile, molte volte, riuscire a capire in che direzione procedere al fine di dare un servizio migliore all'utente finale. Prendiamo ad esempio il rilascio di nuove versioni di Indico: gli sviluppatori al CERN lavorano costantemente per rilasciare versioni sempre aggiornate di Indico. Tuttavia non sempre gli utenti aggiornano Indico all'ultima versione, rimanendo spesso indietro di molte versioni anche per anni. Come scegliere, quindi, il momento migliore per rilasciare una nuova versione? Come fare a capire quando la community è "pronta" ad affrontare un cambiamento del genere? Come decidere se scrivere o meno uno script di migrazione diretto da una versione obsoleta a una più nuova? Tutte queste domande non hanno semplice risposta, soprattutto per via del fatto che gli sviluppatori non hanno modo di conoscere come gli utenti utilizzano Indico. Sarebbe molto più semplice se, ad esempio, si potesse sapere quale versione di Indico utilizzano i vari utenti nel mondo, con che frequenza aggiornano Indico o magari che versione di Python utilizzano per eseguire Indico. Ovvero l'idea di fondo è che ogni aggiornamento ad Indico dev'essere fatto non soltanto dal punto di vista ingegneristico di sviluppo del software ma anche, e soprattutto, dal punto di vista della community: un cambiamento scomodo o non molto ben accetto dalla community risulterà inutile o, peggio ancora, dannoso, per quanto straordinario possa sembrare.

L'idea di questo terzo progetto dell'Indico KT Project era quindi di raccogliere, in forma anonima, tutta una serie di informazioni di utilizzo di ogni istanza, così che poi, formando delle statistiche sui dati raccolti, fosse possibile agli sviluppatori al CERN di prendere decisioni più ponderate e che più rispecchiassero le effettive esigenze della community.



Si è sentito, quindi, il bisogno di sviluppare un software in grado di raccogliere tutti questi dati dalle varie istanze di Indico e di costruire una serie di statistiche da mostrare in forma grafica e intuitiva. Essendo ogni installazione di Indico denominata “istanza”, un software di questo tipo è detto *instance tracker* (ovvero “tracciatore di istanze” in inglese).

Senza entrare nei dettagli dell’implementazione (di cui parleremo in Sezione 6.2), possiamo anticipare che l’instance tracker prodotto in questa fase è formato da una parte centrale, che si occupa dell’archiviazione e dell’elaborazione dei dati raccolti, ed una seconda parte che si occupa invece di raccogliere questi dati da ogni istanza. La seconda parte prevede la specifica di una serie di canali di comunicazione, attraverso i quali il software di tracciamento e ogni singola istanza possono comunicare. Questa descrizione del software di tracciamento è assimilabile a quella di un mollusco, in possesso di un grande cervello centrale che elabora dati e di una serie di tentacoli grazie ai quali cattura gli oggetti esterni. Quest’analogia tra instance tracker e mollusco giustifica il nome scelto, alla fine della sua fase di sviluppo, per il software di tracciamento: Cephalopod, ovvero “Cefalopode” in italiano.

Chiaramente, può succedere che gli amministratori di una specifica istanza decidano di non voler condividere questi dati anonimi con il team di sviluppo al CERN. Per questo Indico è stato adattato a Cephalopod in modo tale da permettere agli amministratori di ogni istanza di decidere se condividere o meno questi dati, con la possibilità di modificare la scelta in ogni momento (come vedremo in Sezione 6.3).

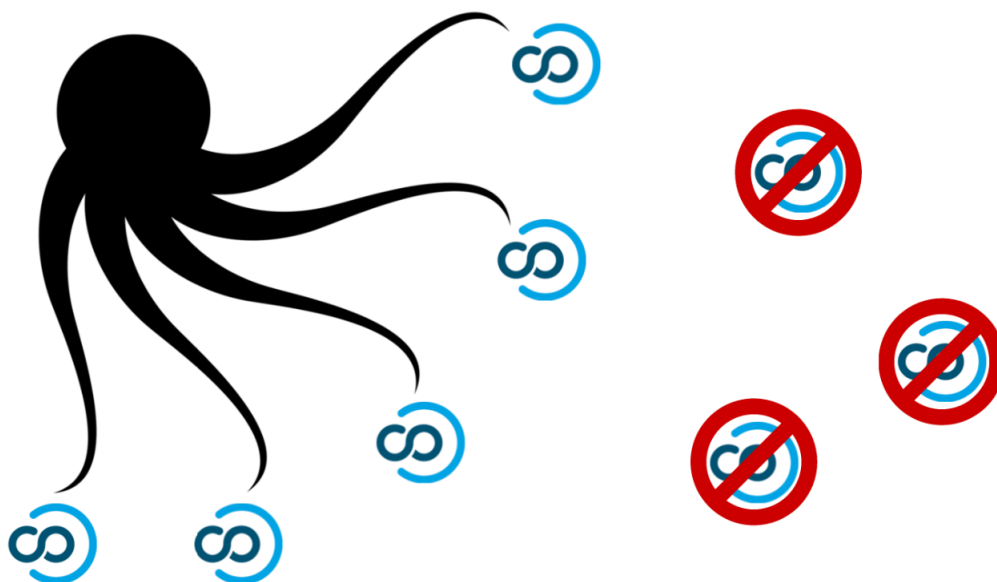


Figura 15: Stilizzazione di Cephalopod (fonte [9]).

In Figura 15 possiamo vedere la stilizzazione dell'applicazione Cephalopod la quale, tramite ogni "tentacolo", raccoglie informazioni da ogni diversa istanza di Indico escludendo, ovviamente, quelle istanze che hanno scelto di non partecipare a quest'iniziativa di raccolta dati.

## 6.1 CARATTERISTICHE PRINCIPALI

Cephalopod è un'applicazione web basata su Flask che permette la registrazione di utenti (ovvero gli amministratori di Cephalopod e idealmente i principali sviluppatori dell'applicazione tracciata). Gli amministratori registrati potranno, tramite una semplice interfaccia web, accedere alla lista delle istanze tracciate, tramite la quale si accede alla pagina di gestione di ogni singola istanza, oppure alla pagina di statistiche, dove vengono rappresentate in forma grafica tutte le informazioni necessarie agli amministratori per prendere decisioni.

In questa Sezione verranno descritte le principali caratteristiche di Cephalopod, nonché alcune istruzioni su come utilizzarlo.

### 6.1.1 Installazione e configurazione

Per installare Cephalopod è necessario, innanzitutto, clonare il repository Github sul quale è archiviato il codice sorgente di Cephalopod e quindi posizionarsi nella cartella principale del codice. Per fare questo basta eseguire i seguenti due comandi da linea di comando:

```
$ git clone https://github.com/indico/cephalopod.git
$ cd cephalopod
```

Una volta clonato con successo il repository del codice sorgente di Cephalopod, si può procedere con l'installazione. Per installare Cephalopod, si hanno due scelte. La prima (consigliata) prevede di installare Cephalopod all'interno di un ambiente virtuale Python dedicato, eseguendo le seguenti istruzioni:

```
$ virtualenv env
$ . env/bin/activate
$ pip install -r requirements.txt
```

Altrimenti si può scegliere di installare Cephalopod direttamente per l'utente corrente. Per fare questo basterà eseguire il seguente comando:

```
$ python setup.py install
```

Una volta terminata l'installazione di Cephalopod tramite una delle due modalità sopra esposte, si deve procedere con alcune fasi di configurazione, prima di poter iniziare ad utilizzare l'applicazione. Cephalopod utilizza un file di configurazione denominato `settings.cfg`, all'interno della cartella `cephalopod/`. Inizialmente il file non è presente e dev'essere creato dall'utente per poter utilizzare il software. Tuttavia, nel codice sorgente, è stato incluso un file di configurazione di esempio, denominato `settings.cfg.example`, sul quale ci si può basare come punto di partenza per configurare Cephalopod. A riguardo, un comando utile potrebbe essere il seguente, che permette di copiare le configurazioni d'esempio nel file di configurazioni vero e proprio:

```
$ cp cephalopod/settings.cfg.example cephalopod/settings.cfg
```

Bisogna tener presente, in generale quando si personalizza il file di configurazione, che esso deve essere scritto rispettando una corretta sintassi Python. Di seguito mostriamo la struttura del file di configurazione d'esempio:

```
# Core settings
BABEL_DEFAULT_TIMEZONE = "UTC"
BABEL_DEFAULT_LOCALE = "en_GB"
ASSETS_DEBUG = False
SECRET_KEY = ''
APP_NAME = 'Cephalopod'
CRAWLING_ENDPOINTS = [
    {
        'url': '/example/endpoint',
        'headers': {'Accept': 'application/json'}
    },
    {
        'url': '/another/example/endpoint'
    }
]
CRAWLED_FIELDS_SETTINGS = {
    'example_field': {
        'label': 'The Example Field!',
        'chart': True,
        'chart_type': 'line',
        'aggregation': 'sum',
        'chart_aggregation': 'avg',
        'chart_aggregate_by': 'another_example_field'
    },
    'example_field_2': {
        'chart': True,
        'chart_type': 'pie',
        'chart_aggregation': 'avg',
```

```

        'chart_aggregate_by': 'country'
    }
}

# Database settings
SQLALCHEMY_DATABASE_URI = 'postgresql:///cephalopod'

# Celery settings
CELERY_BROKER_URL = 'redis://127.0.0.1:6379/0'
from datetime import timedelta
CELERYBEAT_SCHEDULE = {
    'crawl-everything': {
        'task': 'cephalopod.crawler.crawl_all',
        'schedule': timedelta(days=1)
    }
}

```

La prima modifica necessaria al file di configurazione consiste nell'impostare il campo `SECRET_KEY` (utilizzato da Flask): basta aggiungere una parola chiave o una stringa alfanumerica casuale. Dopodiché si possono impostare i campi `BABEL_DEFAULT_TIMEZONE` e `BABEL_DEFAULT_LOCALE` che determinano, rispettivamente, in quale fuso orario e quale formato mostrare l'orario all'interno del software (di default sono `UTC` e `en_GB`, rispettivamente). Infine si può impostare il nome dell'applicazione che vogliamo tracciare (ad esempio Indico) tramite il campo `APP_NAME`.

Riguardo ai restanti campi di configurazione, parleremo dei *crawled field* (traducibili come “campi estratti” in italiano) qui di seguito, mentre tratteremo gli altri campi in Sezione 6.2, quando analizzeremo Cephalopod dal punto di vista implementativo.

### 6.1.2 Non solo Indico

Una delle caratteristiche centrali di Cephalopod è che, sebbene sia stato sviluppato con il fine principale di essere utilizzato per tracciare le istanze di Indico, esso è stato scritto in modo da poter funzionare per una generica applicazione web. Infatti Indico non è la sola applicazione web sviluppata e gestita all'interno del CERN. Un esempio è Invenio<sup>1</sup>, un software, come Indico, open source che gestisce tutta la documentazione informatizzata al CERN e funge da libreria digitale.

Alcuni parametri del file `settings.cfg`, come `APP_NAME` o `CRAWLING_ENDPOINTS`, servono proprio a definire quale applicazione deve tracciare Cephalopod e come deve farlo. `APP_NAME`, ad esempio, serve a definire il nome dell'applicazione

<sup>1</sup> <http://invenio-software.org/>.

tracciata (“Indico”, ad esempio) mentre `CRAWLING_ENDPOINTS` serve a definire come Cephalopod deve comunicare con le istanze da tracciare, come vedremo meglio in Sezione 6.2. L’aggiunta di questo tipo di opzioni ha sicuramente reso lo sviluppo di Cephalopod più complesso ma ha anche permesso a Cephalopod di essere un’applicazione modulare e generale, utilizzabile da una qualsiasi applicazione web. Inoltre, come vedremo successivamente, sarà anche possibile definire dei campi, che Cephalopod potrà estrarre, personalizzati e specifici per il tipo di applicazione tracciata. Indico, ad esempio, potrebbe aver bisogno di estrarre il numero di eventi in una certa istanza, mentre Invenio il numero di documenti presenti nella libreria.

Chiaramente, Cephalopod non può tracciare un’applicazione che non sia stata prima adattata ad essere tracciata da Cephalopod, ovvero a comunicare con esso. In particolare, come vedremo meglio all’interno delle Sezioni 6.2 e 6.3, un’applicazione dovrà implementare degli endpoint, ovvero dei particolari indirizzi tramite i quali Cephalopod può estrarre i dati. In Sezione 6.3 vedremo come questa implementazione è avvenuta nel caso specifico di Indico.

Quindi, ricapitolando, Cephalopod è indirizzato a tutti gli sviluppatori di una generica applicazione web della quale sono interessati tracciarne le istanze ed il loro utilizzo nel mondo. Uno sviluppatore, dopo aver installato e configurato appropriatamente Cephalopod, dovrà quindi occuparsi di implementare gli endpoint sull’applicazione stessa, in modo da permettere la corretta comunicazione tra applicazione e tracciatore.

### 6.1.3 Campi principali e campi estratti

È chiaro a questo punto che la mansione principale di Cephalopod è quella di raccolta dati da ogni singola istanza dell’applicazione tracciata. Risulta lecito a questo punto chiedersi quali informazioni vengano raccolte da Cephalopod. Le informazioni raccolte si dividono in due gruppi: informazioni *principali* ed informazioni *estratte*.

**CAMPI PRINCIPALI** Le informazioni principali sono campi di base che è necessario raccogliere per ogni tipo di applicazione. I campi principali, in particolare, sono 6 e sono i seguenti:

- **Organisation:** contiene il nome dell’organizzazione nella quale è installata l’istanza dell’applicazione;
- **URL:** l’indirizzo alla pagina principale dell’istanza;
- **Contact name:** il nome del responsabile dell’istanza;
- **Email:** l’indirizzo email a cui è possibile raggiungere il responsabile;

- **UUID:** identificatore UUID (Universally Unique IDentifier) assegnato all'istanza;
- **Enabled:** indica se l'istanza ha attivato il servizio di tracciamento oppure no.

Questi campi sono essenziali per Cephalopod ed ogni istanza, se e quando attiva il sistema di tracciamento, dovrà assicurarsi di fornirli tutti.

**CAMPI ESTRATTI** I campi estratti (anche detti “campi aggiuntivi”), invece, sono campi specificati dall'amministratore di Cephalopod e possono essere di svariata natura: per quanto riguarda Indico, ad esempio, si potrebbe pensare a raccogliere il numero di eventi totali, o il numero di utenti registrati, o ancora la versione di Python utilizzata per una determinata istanza o la versione di Indico installata. I campi estratti possono essere di molti tipi: numerici, testuali, booleani, ecc. I campi estratti hanno il fine principale di andare a formare le statistiche, sulla base delle quali l'amministratore potrà fare delle analisi e prendere delle decisioni.

Per comunicare a Cephalopod come trattare i campi estratti (e quanti e quali sono) si utilizza il campo `CRAWLED_FIELDS_SETTINGS` all'interno del file di configurazione `settings.cfg`. In questo senso Cephalopod è un'applicazione modulare in quanto permette all'amministratore di specificare tutti i campi extra che desidera e di personalizzarne il trattamento senza bisogno di dover andare a modificare il codice sorgente di Cephalopod: tutto sarà gestibile dal file di configurazione, tramite il campo `CRAWLED_FIELDS_SETTINGS`.

Entrando più nel dettaglio, il campo `CRAWLED_FIELDS_SETTINGS` dev'essere un dizionario Python, dove ogni elemento dev'essere indicizzato da un nome chiave che identifica il campo, il quale deve coincidere con il nome di un campo estratto dall'istanza (come vedremo meglio in Sezione 6.2): ogni nome che non combacia verrà ignorato.

Per ogni campo aggiuntivo sarà quindi possibile selezionare una serie di opzioni, riassunte qui di seguito:

- `label`: stringa che specifica come rappresentare il nome del campo all'interno di Cephalopod; se non specificato, il valore di default è dato dal nome del campo sostituendo uno spazio ad ogni simbolo di underscore (`_`) e rendendo maiuscola la prima lettera della prima parola;
- `chart`: valore booleano che specifica se quel campo verrà incluso o meno tra le statistiche; default `False`;
- `chart_type`: se `chart` è impostato a `True`, allora specifica che tipo di grafico utilizzare per visualizzare le statistiche relative al campo; i valori accettati sono `'bar'`, per avere un grafico a barre verticali, `'line'`, per un grafico a linea, e `'pie'`, per un grafico a torta; default `'bar'`;

- `aggregation` : se il campo non può essere visualizzato direttamente (ad esempio è una lista o un dizionario), si deve definire una funzione di aggregazione per poter associare un singolo valore al campo; al momento sono supportati soltanto campi numerici di questo tipo; le funzioni di aggregazione possono essere `None`, se nessuna aggregazione è necessaria, `'sum'`, per restituire la somma, `'avg'`, per restituire il valor medio, e `'min'` e `'max'`, per restituire, rispettivamente, valor minimo e massimo; default `None`;
- `chart_aggregation` : opzione simile a `aggregation`, serve a specificare come aggregare i valori del campo per ogni categoria quando `chart` è `True`; valori possibili sono `'count'`, per mostrare il numero di istanze per ogni categoria, `'sum'`, `'avg'`, `'min'` e `'max'`; default `'count'`;
- `chart_aggregate_by` : stringa che specifica rispetto a quale altro campo raggruppare il campo corrente nelle statistiche; i valori accettati sono il nome di uno qualsiasi degli altri campi estratti oppure `'country'`, per raggruppare i valori rispetto al paese dell'istanza; se `chart_aggregation` è impostato a `'count'`, allora questa opzione verrà ignorata; default `'country'`.

Mostriamo di seguito, un paio di esempi per rendere più chiaro come funziona la definizione di campi aggiuntivi. Come primo esempio prendiamo il caso di Indico e supponiamo di voler definire un campo che contiene la versione di Python utilizzata da ogni istanza. Chiamiamo questo campo `python_version`. Ci aspettiamo che i valori di questo campo siano stringhe, quindi non c'è bisogno di specificare alcuna funzione di aggregazione. Supponiamo poi di voler includere questo campo tra le statistiche, mostrando un grafico a barre che, per ogni versione Python utilizzata, indica quante istanze ne fanno uso. Quindi, seguendo le definizioni date prima, dovremmo avere `chart` messo a `True`, `chart_type` settato su `'bar'` (o non incluso, essendo `'bar'` il valore di default) e `chart_aggregation` impostato a `'count'`. Il risultato è il seguente frammento di codice Python del file di configurazione:

```
CRAWLED_FIELDS_SETTINGS = {
    'python_version': {
        'label': 'Python version',
        'chart': True,
        'chart_type': 'bar',
        'chart_aggregation': 'count'
    }
}
```

Se, quindi, volessimo rappresentare, tramite grafico a linea, anche il numero medio di eventi totali in ogni istanza che utilizza una determinata versione di Python, si potrebbe utilizzare la seguente configurazione:

```
CRAWLED_FIELDS_SETTINGS = {
    'python_version': {
        'label': 'Python version',
        'chart': True,
        'chart_type': 'bar',
        'chart_aggregation': 'count'
    },
    'events': {
        'label': 'Events',
        'chart': True,
        'aggregation': 'sum',
        'chart_type': 'line',
        'chart_aggregation': 'avg',
        'chart_aggregate_by': 'python_version'
    }
}
```

In quest'ultimo esempio, si aggregano gli eventi tramite somma in quanto Indico raccoglie il numero di eventi diviso per categoria (quindi avremo un dizionario) e si richiede il numero totale di eventi. Impostando invece `chart_aggregation` a `'avg'` indichiamo a Cephalopod di mostrare, per ogni versione Python, il numero medio di eventi (totali), ovvero di fare la media tra il numero di eventi totali di ogni istanza che utilizza la stessa versione di Python.

Per avere un'idea di come potrebbero apparire queste due configurazioni d'esempio, si veda la Figura 21 a pagina 60.

#### 6.1.4 Interfaccia web

Parliamo adesso di com'è strutturata l'interfaccia web tramite la quale gli amministratori utilizzano Cephalopod. La struttura dell'applicazione è molto semplice ed è divisa, principalmente, in 4 sezioni: la pagina principale, la lista delle istanze, la pagina di gestione e dei dettagli di un'istanza e la pagina delle statistiche.

**PAGINA PRINCIPALE E DETTAGLI** La pagina principale viene utilizzata soltanto per effettuare il login e per accedere alle altre sezioni dell'applicazione. Per il resto, la pagina è piuttosto vuota e non presenta particolari funzionalità. Infatti è stata pensata come semplice punto di appoggio per eseguire il login e poi spostarsi sulle altre sezioni.

Degna di nota sono anche la barra principale e le *breadcrumbs* ("briciole di pane" in italiano). La barra principale, posta in alto, è una barra reattiva (*responsive*, in



inglese), ovvero che si adatta alla larghezza dello schermo e, se troppo piccolo, si trasforma in un pulsante tramite il quale le varie voci vengono visualizzate in un apposito menu a tendina. Sulla destra della barra è stato posizionato il nome dell'utente attualmente loggato ed un pulsante per effettuare il logout. Le breadcrumbs, invece, sono poste subito sotto la barra del menu e permettono di sapere in ogni momento in che posizione dell'applicazione ci troviamo e di poter navigare velocemente da una pagina all'altra. Per garantire una maggior semplicità di navigazione, sia la barra del menu che le breadcrumbs sono state fissate sempre in alto in primo piano, sia cambiando pagina che scorrendo verso il basso. In Figura 16 si può vedere come appaiono la barra del menu e le breadcrumbs.



Figura 16: Barra del menu e breadcrumbs in Cephalopod.

**LISTA DELLE ISTANZE** La pagina della lista delle istanze registrate, denominata *server list*, serve invece a fornire a colpo d'occhio la situazione delle istanze attive al momento. In questa pagina si trova una lista delle istanze registrate, al momento o in passato, posizionata al centro. Per ogni voce sono indicati il nome dell'istanza, l'indirizzo e la data dell'ultima volta in cui sono stati aggiornati i dati. In Figura 17 possiamo vedere un esempio della pagina della lista delle istanze<sup>1</sup>.

Possiamo notare che per ogni istanza sono presenti tre pulsanti: il primo, indicato da una clessidra, serve ad aprire una nuova pagina, dedicata alla gestione di quell'istanza; il secondo, indicato dal simbolo di aggiornamento (o refresh), avvia il processo di aggiornamento dei dati (principali e aggiuntivi) per quella particolare istanza; il terzo invece, indicato da una croce, permette di eliminare quell'istanza dalla lista. La freccia alla sinistra del nome di ogni istanza serve ad espandere i dettagli di quell'istanza, aprendo un riquadro a tendina sotto alla voce, dove vengono riassunti i valori di tutti gli altri campi non mostrati. In alto si possono notare tre bottoni: il primo ("Crawl all") serve a forzare l'aggiornamento dei dati per tutte le istanze attive; il secondo ("Show") apre un menu a tendina tramite il quale si possono selezionare quali istanze mostrare nella lista (attive, disattive o entrambe); il terzo ("Additional data") serve invece ad includere direttamente su ogni riga i campi non mostrati inizialmente (senza bisogno di cliccare sulla freccia a sinistra di ogni voce), come possiamo vedere in Figura 18.

<sup>1</sup> Si faccia attenzione che in quest'immagine, come in tutte le altre presenti all'interno di questo capitolo, i dati presenti sono del tutto fittizi e creati ad-hoc al solo scopo illustrativo, in quanto Cephalopod non è ancora stato rilasciato e non è ancora in utilizzo, né al CERN né altrove.

Indico Tracker / Servers / Server list

## List of servers

Number of active instances: 11/12

[Crawl all](#) [Show](#) [Additional data](#)

Search:

Organisation name	URL	Last crawled	
➤ CERN	<a href="http://pcituds06.cern.ch:8000">http://pcituds06.cern.ch:8000</a>	16 Jun 2014 08:48:51 (Europe/Zurich)	<a href="#">Q</a> <a href="#">C</a> <a href="#">X</a>
➤ CERN	<a href="http://pcituds06.cern.ch:8000">http://pcituds06.cern.ch:8000</a>	16 Jun 2014 08:48:52 (Europe/Zurich)	<a href="#">Q</a> <a href="#">C</a> <a href="#">X</a>
➤ CERN	<a href="http://pcituds06.cern.ch:8000">http://pcituds06.cern.ch:8000</a>	16 Jun 2014 08:48:53 (Europe/Zurich)	<a href="#">Q</a> <a href="#">C</a> <a href="#">X</a>
➤ CERN	<a href="http://pcituds06.cern.ch:8000">http://pcituds06.cern.ch:8000</a>	16 Jun 2014 08:48:53 (Europe/Zurich)	<a href="#">Q</a> <a href="#">C</a> <a href="#">X</a>
➤ CERN	<a href="http://pcituds06.cern.ch:8000">http://pcituds06.cern.ch:8000</a>	16 Jun 2014 08:48:54 (Europe/Zurich)	<a href="#">Q</a> <a href="#">C</a> <a href="#">X</a>
➤ Tom's Virtual Indico	<a href="http://pcituds06.cern.ch:8000">http://pcituds06.cern.ch:8000</a>	16 Jun 2014 08:48:54 (Europe/Zurich)	<a href="#">Q</a> <a href="#">C</a> <a href="#">X</a>
➤ Google	<a href="http://www.google.it">www.google.it</a>	Never	<a href="#">Q</a> <a href="#">C</a> <a href="#">X</a>
➤ Youtube	<a href="https://www.youtube.com">https://www.youtube.com</a>	Never	<a href="#">Q</a> <a href="#">C</a> <a href="#">X</a>
➤ ThePirateBay	<a href="http://thepiratebay.se">http://thepiratebay.se</a>	Never	<a href="#">Q</a> <a href="#">C</a> <a href="#">X</a>
➤ Amazon.uk	<a href="http://www.amazon.co.uk">http://www.amazon.co.uk</a>	Never	<a href="#">Q</a> <a href="#">C</a> <a href="#">X</a>
➤ Imgsrc	<a href="http://imgsrc.ru">http://imgsrc.ru</a>	Never	<a href="#">Q</a> <a href="#">C</a> <a href="#">X</a>

Figura 17: Pagina della lista delle istanze in Cephalopod.

Infine notiamo una barra di ricerca, in alto a destra, che permette di filtrare le istanze della lista in base a delle parole chiave, e delle piccole frecce accanto ad ogni colonna della lista, che permettono di ordinare la lista secondo quel determinato campo (ad esempio un amministratore potrebbe voler visualizzare la lista per ordine decrescente di numero totale di eventi).

**DETTAGLI DI UN'ISTANZA** Se, dalla lista delle istanze, si clicca sul pulsante contrassegnato da una lente di ingrandimento, si accede ad una pagina dedicata ai dettagli di quell'istanza. In Figura 19 possiamo vedere una schermata rappresentativa di come appare la pagina dei dettagli.

Analizziamo, adesso, la struttura della pagina. In alto troviamo l'indirizzo base dell'istanza, in qualità di titolo della pagina, e due bottoni di azione: "Crawl now", che forza la raccolta di informazioni per quell'istanza, e "Delete server", che invece la elimina dal database di Cephalopod.

Subito sotto alle azioni troviamo due riquadri: "Main information" e "Crawled information". Il riquadro "Main information" raccoglie tutti i campi principali descritti prima, mostrandone il valore relativo a quell'istanza particolare. Il pulsante simile ad una penna che scrive posizionato accanto al titolo "Main information" serve per permettere all'amministratore di Cephalopod di modificare i campi principali per una determinata istanza, nel caso avessero bisogno di

Indico Tracker Home Servers Statistics Tommaso Log out

Indico Tracker / Servers / Server list

### List of servers

Number of active instances: 12/13

Crawl all Show Additional data

Contributions (sum)  
Events (sum)  
Indico version  
Language  
Python version  
Resources  
Updated  
Users

Search:

Organisation name	URL	Last crawled
CERN	<a href="http://pctiuds06.cern.ch:8000">http://pctiuds06.cern.ch:8000</a>	16 Jun 2014 08:48:51 (Europe/Zurich)
CERN	<a href="http://pctiuds06.cern.ch:8000">http://pctiuds06.cern.ch:8000</a>	16 Jun 2014 08:48:52 (Europe/Zurich)
CERN	<a href="http://pctiuds06.cern.ch:8000">http://pctiuds06.cern.ch:8000</a>	16 Jun 2014 08:48:53 (Europe/Zurich)
CERN	<a href="http://pctiuds06.cern.ch:8000">http://pctiuds06.cern.ch:8000</a>	16 Jun 2014 08:48:53 (Europe/Zurich)
CERN	<a href="http://pctiuds06.cern.ch:8000">http://pctiuds06.cern.ch:8000</a>	16 Jun 2014 08:48:54 (Europe/Zurich)
Tom's Virtual Indico	<a href="http://pctiuds06.cern.ch:8000">http://pctiuds06.cern.ch:8000</a>	16 Jun 2014 08:48:54 (Europe/Zurich)
Google	<a href="http://www.google.it">www.google.it</a>	Never
Youtube	<a href="https://www.youtube.com">https://www.youtube.com</a>	Never
ThePirateBay	<a href="http://thepiratebay.se">http://thepiratebay.se</a>	Never
Amazon.uk	<a href="http://www.amazon.co.uk">http://www.amazon.co.uk</a>	Never
Imgsr	<a href="http://imgsrc.ru">http://imgsrc.ru</a>	Never
Tom's virtual Indico	<a href="http://pctiuds06.cern.ch:8000">http://pctiuds06.cern.ch:8000</a>	Never

Figura 18: Pagina della lista delle istanze in Cephalopod col menu a tendina “Additional data” aperto.

essere aggiornati: cliccandoci, il valore di ogni campo si trasformerà in un input field, dove sarà possibile inserire il valore aggiornato, oppure lasciarlo invariato, per poi cliccare su un bottone di salvataggio che comparirà durante la modifica. Nel riquadro “Crawled information” vengono invece raccolti tutti i dati estratti aggiuntivi, ovvero i campi definiti dall’amministratore di Cephalopod all’interno del file `settings.cfg`. In fondo al riquadro, inoltre, verrà visualizzata l’ultima data in cui sono stati estratti i dati. Come possiamo vedere dall’esempio in Figura 19, alcuni di questi dati possono essere di tipo numerico (numero di utenti, numero di risorse, ecc...) oppure di tipo testuale (come la lingua o la data di ultimo aggiornamento di Indico). Si può vedere inoltre che i dati aggregati, come il numero di eventi, sono affiancati, tra parentesi, dalla funzione di aggregazione scelta (nell’esempio abbiamo il numero totale di eventi, in quanto l’aggregazione è fatta tramite somma).

Infine, in basso compare una mappa, che mostra la geolocalizzazione dell’istanza. Come vedremo meglio in Sezione 6.2, la geolocalizzazione avviene in automatico sfruttando l’indirizzo IP (Internet Protocol) durante la fase di estrazione dei dati.

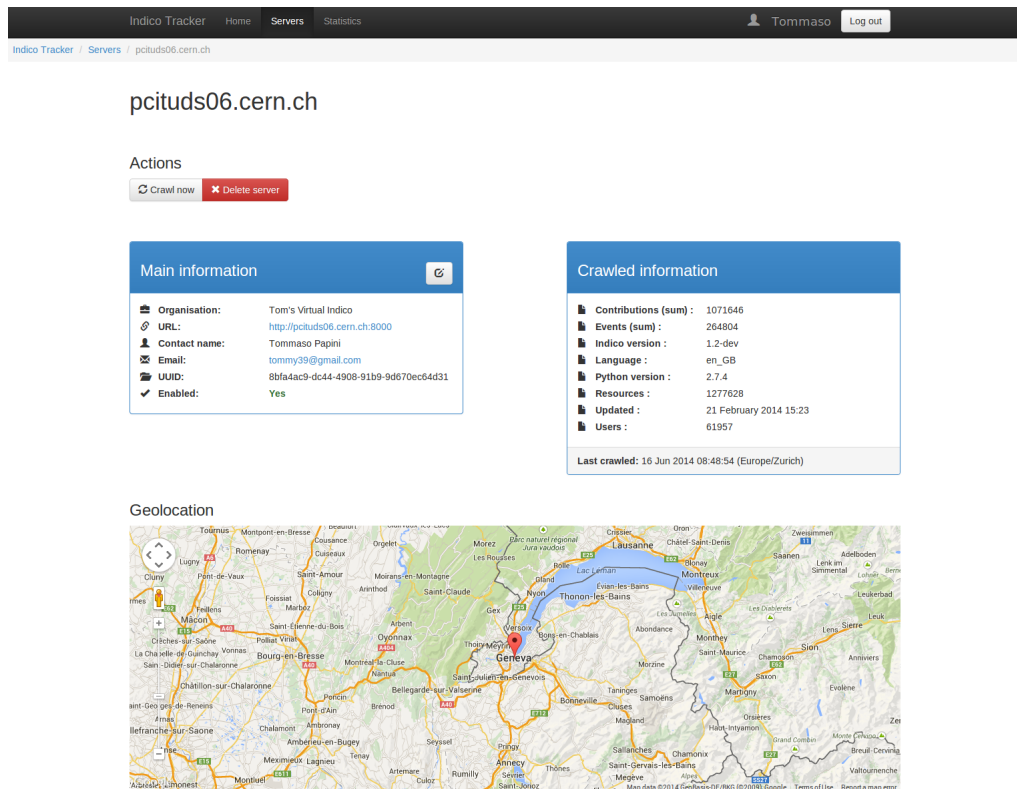


Figura 19: Pagina dei dettagli e di gestione di un'istanza.

**STATISTICHE** La pagina delle statistiche è forse la pagina più importante per l'amministratore di Cephalopod in quanto raccoglie tutti i dati ottenuti dalle varie istanze e li presenta all'amministratore in forma grafica ed intuitiva, permettendo, a colpo d'occhio, di qual è la situazione delle istanze rispetto a certi aspetti di interesse.

La prima parte della pagina è fissa ed è illustrata in Figura 20.

Nella parte superiore troviamo una mappa ed un grafico a torta che rappresentano la distribuzione delle istanze per paese. Nella mappa compare un puntino in corrispondenza di ogni istanza registrata e geolocalizzata correttamente e ogni stato viene colorato di una determinata sfumatura di blu oppure grigio, dove blu scuro indica un'alta concentrazione di istanze in quello stato, blu chiaro una bassa concentrazione e grigio indica nessuna istanza in quel determinato stato. Il grafico a torta serve invece a mostrare, a colpo d'occhio, la percentuale di istanze per ogni stato.

Subito sotto alla distribuzione per paese, viene mostrato un grafico a linea dove si può vedere come si evolve il numero di istanze registrate a Cephalopod nel tempo.



Figura 20: Parte principale della pagina delle statistiche.

I grafici della distribuzione per paese e delle istanze registrate nel tempo sono sempre presenti nella pagina delle statistiche e non possono essere modificati. Subito sotto, invece, si sviluppa la parte dinamica della pagina, ovvero la parte dedicata alle statistiche definite dall'amministratore. In particolare, tutti i grafici che l'amministratore ha abilitato e configurato in `settings.cfg` saranno visualizzati in questa pagina, in ordine di definizione, immediatamente sotto al grafico delle istanze registrate.

A titolo d'esempio mostriamo alcuni grafici, in Figura 21, che mostrano diversi modi di definire delle statistiche sui vari campi aggiuntivi.

In particolare, il primo ed il terzo grafico (partendo dall'alto a sinistra) in Figura 21 sono il risultato delle configurazioni d'esempio viste all'interno della Sottosezione 6.1.3.

### 6.1.5 Scheduler

Cephalopod utilizza Celery come scheduler, ovvero come modulo che si occupa di gestire eventi periodici. Le impostazioni di Celery sono controllate interamente dal file di configurazioni `settings.cfg`. Se, ad esempio, l'amministratore di Cephalopod volesse impostare l'estrazione dei dati da tutte le istanze in modo

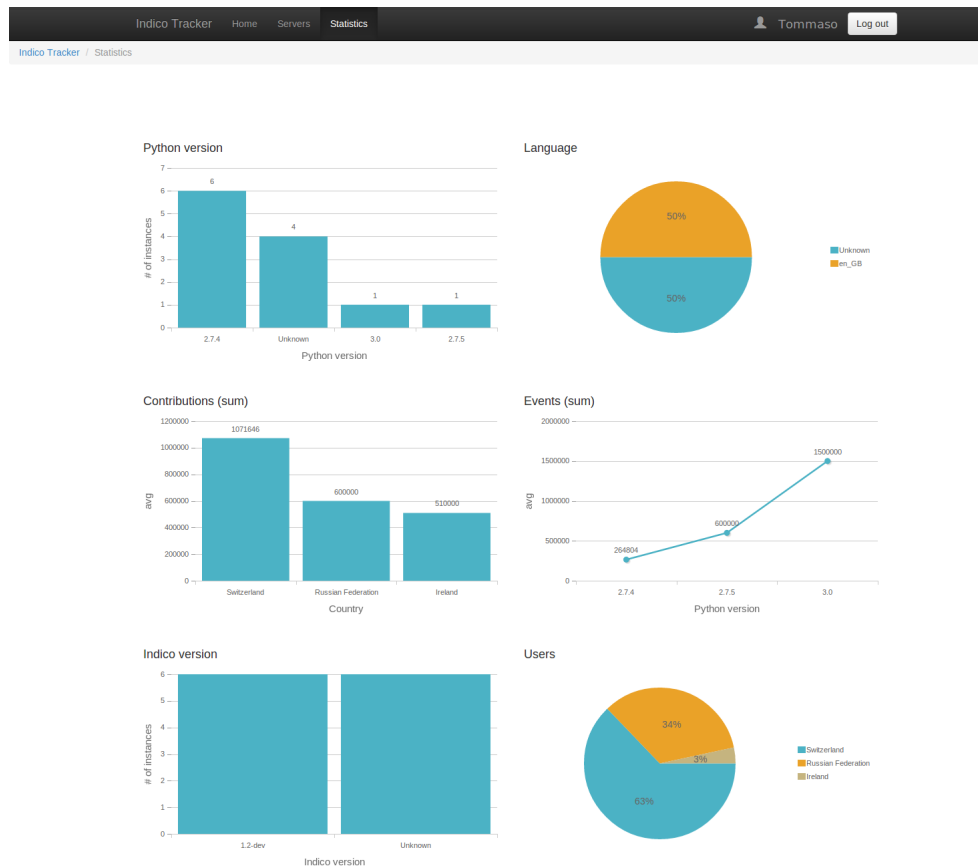


Figura 21: Parte personalizzabile della pagina delle statistiche che contiene le statistiche definite dall'amministratore.

che venga eseguita in automatico ogni giorno, basterebbe includere il seguente frammento di codice all'interno di `settings.cfg` :

```
from datetime import timedelta
CELERYBEAT_SCHEDULE = {
    'crawl-everything': {
        'task': 'cephalopod.crawler.crawl_all',
        'schedule': timedelta(days=1)
    }
}
```

Il campo `task` serve a indicare che comando eseguire e dev'essere una qualsiasi funzione messa a disposizione da Cephalopod (in questo caso si richiama la funzione `crawl_all` del file `crawler.py` all'interno della cartella `cephalopod`). Nel

campo `schedule` si deve invece indicare l'intervallo di tempo che deve passare tra due esecuzioni successive dello stesso comando (in questo esempio, un giorno).

#### 6.1.6 Script di gestione

Abbiamo già mostrato come sia possibile, per un amministratore di Cephalopod, gestire e aggiornare le istanze tramite l'applicazione web stessa, o come sia possibile gestire le impostazioni, ad esempio delle statistiche, attraverso il file di configurazione `settings.cfg`. Ci sono tuttavia alcune operazioni avanzate che, con questi due soli strumenti, non è possibile effettuare. Per dare all'amministratore, quindi, tutti gli strumenti necessari a gestire Cephalopod nel migliore dei modi è stato scritto uno script in python che esegue alcune operazioni da linea di comando.

Per utilizzare lo script basta immettere, da linea di comando, il seguente codice, nel caso in cui si sia installato Cephalopod tramite `setup.py`:

```
$ cephalopod <argomenti>
```

Altrimenti, basterà eseguire il seguente comando:

```
$ python manage.py <argomenti>
```

Gli argomenti che si possono passare allo script di gestione sono elencati di seguito:

- `shell` : apre una shell Python interattiva all'interno del contesto dell'applicazione Flask;
- `runserver` : avvia l'applicazione Flask che gestisce il web server di Cephalopod;
- `db drop` : elimina tutte le tabelle del database;
- `db create` : crea le tabelle del database;
- `db recreate` : elimina tutte le tabelle del database e ne crea di nuove (equivalente a fare `drop` e `create`);
- `crawl [uuid]` : estrae i dati da una specifica istanza (se viene passato un UUID), altrimenti tutte;
- `create_usr usr pswd` : crea un nuovo utente con `usr` come username e `pswd` come password di accesso;
- `runworker [concurrency]`: avvia un processo Celery; `concurrency` è un intero che indica con che grado di parallelismo far partire il processo.

## 6.2 DETTAGLI IMPLEMENTATIVI

Dopo aver descritto le caratteristiche principali di Cephalopod, tra cui la sua struttura ed il suo utilizzo, entriamo adesso un po' più nel dettaglio, esaminando gli aspetti tecnici e implementativi del codice di Cephalopod.

Come già detto, Cephalopod è un'applicazione web scritta in Python, basata su Flask e che utilizza elementi HTML (HyperText Markup Language) forniti da Twitter Bootstrap per ottenere un design moderno e reattivo.

### 6.2.1 *Struttura del codice*

I file principali del codice sorgente di Cephalopod sono tutti contenuti all'interno della cartella `cephalopod/`. I file più importanti che troviamo in questa cartella sono `settings.cfg`, `manage.py`, `factory.py` e `crawler.py`.

Del file di configurazione `settings.cfg` abbiamo già ampiamente parlato nella Sezione precedente. Il file `manager.py` è lo script di gestione di cui abbiamo parlato prima e contiene la definizione di tutti i comandi possibili descritti prima. `factory.py` serve a definire le funzioni che istanziano una nuova applicazione Flask (quando si avvia il web server) o un nuovo processo Celery. Infine, `crawler.py` definisce tutte le funzioni dedicate all'estrazione delle informazioni dalle istanze, come ad esempio la geolocalizzazione.

Il codice di Cephalopod si sviluppa quindi all'interno di quattro cartelle principali: `api/`, `models/`, `templates/` e `webinterface/`, sempre all'interno della cartella `cephalopod/`.

La cartella `api/` contiene il solo file `instance.py`, dove vengono definite le API messe a disposizione di Cephalopod. All'interno di `models/` troviamo i due file, `instance.py` e `user.py`, che definiscono il tipo di oggetti che possiamo memorizzare all'interno del database. Dentro `templates/` troviamo invece una serie di file, in formato `.html`, che servono, appunto, da template per generare pagine dal contenuto dinamico. Infine la cartella `webinterface/` contiene il solo file `misc.py` dove si definiscono le varie pagine di Cephalopod, ovvero la struttura dell'applicazione.

### 6.2.2 *Il Database*

Il database utilizzato da Cephalopod è PostgreSQL. Per poter utilizzare PostgreSQL all'interno di Cephalopod, che è un'applicazione scritta in Python, si è utilizzato il toolkit SQLAlchemy, che mette a disposizione molte funzioni utili per gestire il database.

All'interno della cartella `models` abbiamo i due file, `instance.py` e `user.py`, che definiscono il tipo di oggetti che si possono archiviare all'interno del database.



`instance.py` definisce una classe `Instance` che, intuitivamente, serve a memorizzare le informazioni di una specifica istanza, mentre `user.py` definisce la classe `User`, che memorizza le informazioni di ogni utente.

I campi della classe `Instance` sono:

- `id` : ID da utilizzare come chiave primaria;
- `uuid` : identificatore UUID assegnato all'istanza;
- `enabled` : valore booleano che indica se l'istanza è abilitata o meno;
- `url` : l'URL (Uniform Resource Locator) dell'istanza;
- `contact` : il nome della persona di riferimento;
- `email` : l'indirizzo email della persona di riferimento;
- `organisation` : nome dell'organizzazione nella quale è installata l'istanza;
- `crawl_date` : data dell'ultima estrazione di informazioni;
- `crawled_data` : i dati aggiuntivi raccolti;
- `geolocation` : informazioni sulla geolocalizzazione del server;
- `registration_date` : data di registrazione dell'istanza su Cephalopod.

I campi della classe `User` sono invece:

- `id` : come prima, rappresenta la chiave primaria nel database;
- `username` : username dell'utente;
- `password` : password scelta, memorizzata previa cifratura;
- `registered_on` : data di registrazione dell'utente.

La definizione della classe `User` è, escluse le funzioni accessorie, la seguente:

```
class User(db.Model):
    __tablename__ = "users"
    id = db.Column('user_id', db.Integer, primary_key=True)
    username = db.Column('username', db.String, unique=True, index=True)
    password = db.Column('password', db.String)
    registered_on = db.Column('registered_on', db.DateTime)
```

SQLAlchemy mette a disposizione molti tipi di dato tra cui scegliere, come `db.String` per le stringhe o `db.DateTime` per le date. Tuttavia può capitare di dover includere un campo di un tipo non supportato, nel qual caso è necessario definire prima il tipo di dato e come esso si comporta. In Cephalopod questo è stato necessario per i campi `crawled_data` e `geolocation` della classe `Instance` in quanto essi devono memorizzare strutture dati complesse, in particolare dizionari di tipo JSON (JavaScript Object Notation), e questo tipo non viene offerto da SQLAlchemy. Per aggiungere la possibilità di memorizzare dizionari JSON all'interno del database, si è quindi aggiunta la seguente definizione (fonte [25]):

```
class JSONEncodedDict(TypeDecorator):

    impl = VARCHAR

    def process_bind_param(self, value, dialect):
        if value is not None:
            value = json.dumps(value)
        return value

    def process_result_value(self, value, dialect):
        if value is not None:
            value = json.loads(value)
        return value
```

Quindi, la definizione dei campi `crawled_data` e `geolocation` risulta essere la seguente:

```
crawled_data = db.Column(JSONEncodedDict)
geolocation = db.Column(JSONEncodedDict)
```

SQLAlchemy mette a disposizione anche altre funzioni molto utili, ad esempio `db.drop_all()`, che elimina tutte le tabelle presenti, `db.create_all()`, che crea le tabelle, e `db.session.commit()`, che aggiorna il database con le modifiche fatte.

### 6.2.3 Endpoint ed API

Parliamo adesso di endpoint e di API. Di fatto, quando parliamo di applicazioni web, endpoint ed API significano la stessa cosa. All'interno di Cephalopod, tuttavia, questi due termini vengono utilizzati per esprimere due cose distinte. Con endpoint si intende gli URL che l'applicazione che si vuole tracciare tramite Cephalopod deve implementare per permettere a Cephalopod di estrarre i dati voluti. Con API si intendono invece gli URL messi a disposizione da Cephalopod

per permettere alle varie istanze di comunicare con Cephalopod. Detto questo, URL endpoint ed API esprimono entrambi il concetto di URL tramite il quale si accede a un servizio.

Gli endpoint tramite i quali Cephalopod può accedere ai dati di una certa istanza devono essere definiti in `settings.cfg` ed in particolare nel campo `CRAWLING_ENDPOINTS`. `CRAWLING_ENDPOINTS` è una lista dove ogni elemento corrisponde ad un endpoint. Ogni elemento della lista, a sua volta, dovrà essere un dizionario Python contenente il campo `url`, dove si specifica l'endpoint, ed eventualmente un campo `headers`, che deve contenere un dizionario Python con le opzioni da utilizzare quando si estraggono informazioni da quello specifico endpoint. La stringa specificata nel campo `url` viene concatenata in fondo all'URL di base di ogni istanza per formare l'URL vero e proprio da cui estrarre i dati. Per questo motivo viene chiamato "endpoint": in quanto si specifica soltanto la parte finale dell'URL.

Ovviamente ogni applicazione che si vuole tracciare con Cephalopod deve implementare questi endpoint, per permettere a Cephalopod di comunicare con essa. Questo, chiaramente, dipende da applicazione a applicazione ed ogni sviluppatore ha il compito di implementare le proprie API per permettere alla propria applicazione di comunicare con Cephalopod. In Sezione 6.3 vedremo l'implementazione di questi endpoint nel caso di Indico.

Le API messe a disposizione da Cephalopod per permettere alle istanze di comunicare con lui definiscono invece tre endpoint distinti, ognuno con una diversa funzione. Ogni API è caratterizzata, quindi, da un endpoint, che rappresenta la parte finale dell'URL al quale si può raggiungere Cephalopod, da un tipo di richiesta HTTP (HyperText Transfer Protocol), da dei dati da inviare con la richiesta e da una risposta che si può ricevere dopo aver inviato la richiesta. Di seguito si descrivono le tre API fornite:

- **Create instance:** questa API è utilizzata per inserire una nuova istanza nel database di Cephalopod.
  - *Endpoint:* `/instance/` ;
  - *Tipo di richiesta:* `POST` ;
  - *Dati:* l'URL del server dell'istanza, nome ed email della persona di riferimento e nome dell'organizzazione;
  - *Risposta:* il nuovo UUID assegnato all'istanza.
- **Update instance:** utilizzata per aggiornare i dati dell'istanza ogni qualvolta questi venissero cambiati (include il caso in cui si l'amministratore dell'istanza decida di abilitare/disabilitare il servizio di tracciamento).
  - *Endpoint:* `/instance/<uuid>` ;
  - *Tipo di richiesta:* `PATCH` ;

- *Dati*: uno o più dei seguenti campi: URL dell’istanza, nome o email della persona di riferimento, il nome dell’organizzazione e stato del sistema di tracciamento ( `True` o `False` );
- *Risposta*: un riepilogo dei dati principali dell’istanza in formato JSON.
- **Get instance**: utilizzata per ottenere i dati di una determinata istanza.
  - *Endpoint*: `/instance/<uuid>` ;
  - *Tipo di richiesta*: `GET` ;
  - *Dati*: nessuno;
  - *Risposta*: un riepilogo dei dati principali dell’istanza in formato JSON (come per l’API di aggiornamento).

L’applicazione che si desidera tracciare dovrà quindi fare uso, al suo interno, di queste tre API per comunicare con Cephalopod. Ad esempio l’API di creazione verrà utilizzata non appena si attiva il servizio di tracciamento per una specifica istanza per la prima volta, ad esempio quando l’istanza viene installata. Come utilizzare le API fornite da Cephalopod rimane, tuttavia, a discrezione degli sviluppatori dell’applicazione: per Indico, ad esempio, si è optato per chiedere esplicitamente agli amministratori di ogni istanza di scegliere se attivare o meno il sistema di tracciamento, mentre gli sviluppatori di un’altra applicazione potrebbero scegliere di tenerlo attivo sempre senza possibilità di disattivazione.

#### 6.2.4 Interfaccia web e applicazione

Il codice che risiede dietro le quinte dell’interfaccia web di Cephalopod è diviso in due parti. La prima parte è rappresentata al file `misc.py`, nella cartella `webinterface/`, che contiene tutte le funzioni Python relative ad ogni URL dell’applicazione. La seconda è invece rappresentata dai file contenuti nella cartella `templates/`, ovvero dai template delle pagine in formato HTML.

All’interno di `misc.py` viene definita una serie di funzioni, dove ogni funzione corrisponde ad una determinata azione la quale, a sua volta, è associata ad uno specifico URL. In questo caso non si parla di API in quanto, concettualmente, le API sono raggiungibili dall’esterno, mentre questi URL sono pensati per essere raggiunti soltanto tramite Cephalopod.

L’azione di associare un URL ad una funzione è detto *URL routing* (ovvero “indirizzamento degli URL”) ed è resa disponibile grazie al decoratore di Flask `route()`. Una volta associato l’URL alla funzione desiderata, la funzione può eseguire i comandi specificati ogni volta che si accede a tale URL. In particolare, se l’URL corrisponde alla visualizzazione di una pagina, allora tale funzione dovrà occuparsi di generare la pagina richiesta e quindi restituirla, utilizzando la funzione Flask `render_template()`. Inoltre, Flask mette a disposizione altre

funzioni utili, come ad esempio la funzione `redirect()`, che serve a reindirizzare l'applicazione su un altro URL, oppure il decoratore `login_required`, che permette di eseguire i comandi di una certa funzione se e solo se si chi richiede l'URL è un utente correttamente registrato e loggato, altrimenti restituisce un errore. Di seguito mostriamo due funzioni, a scopo illustrativo, relative alla pagina della lista delle istanze e all'eliminazione di un'istanza:

```
@bp.route('/servers')
@menu('server_list')
@login_required
@breadcrumb('Servers', 'server_list')
def server_list():
    g.breadcrumbs.append(make_breadcrumb('Server list'))
    server_list = Instance.query.all()
    active_instances = sum(1 for server in server_list if server.enabled)
    wvars = {
        'server_list': server_list,
        'extra_fields': get_extra_fields(server_list),
        'active_instances': active_instances
    }
    return render_template('server_list.html', **wvars)

@bp.route('/servers/<id>', methods=('DELETE',))
@login_required
def remove_server(id):
    instance = Instance.query.filter_by(id=id).first()
    db.session.delete(instance)
    db.session.commit()
    return jsonify()
```

Quando l'applicazione deve generare una nuova pagina HTML, lo fa a partire da una serie di parametri e da un determinato template, sfruttando la funzione Flask `render_template()`. Tutti i template disponibili risiedono nella cartella `templates`. I template disponibili sono quattro e corrispondono alle quattro possibili pagine di Cephalopod: `index.html`, `server_list.html`, `manage_server.html` e `statistics.html`. Sono inoltre stati definiti due ulteriori template, `_form.html` e `_layout.html`, che hanno lo scopo di definire gli elementi comuni a tutte le pagine, come ad esempio le breadcrumbs o la barra dei menu. Tutti i template sono stati scritti in Jinja2, descritto in Sezione 3.2.

Infine, lo stile di formattazione di tutte le pagine di Cephalopod è basato su Twitter Bootstrap, in quanto esso fornisce uno stile moderno, minimalista ed interattivo. Per utilizzare gli stili di Twitter Bootstrap all'interno di Cephalopod è bastato importare i file corrispondenti, scaricabili da sito di Bootstrap, all'interno

del template `_layout.html` che, come detto prima, viene utilizzato da tutte le pagine come template di base.

### 6.2.5 Javascript

Essendo un'applicazione web dinamica, Cephalopod fa un largo uso di codice Javascript per gestire tutti gli elementi dinamici e interattivi delle varie pagine dell'applicazione. Senza entrare troppo nel dettaglio, descriviamo soltanto alcune parti più importanti.

Tutti i grafici utilizzati nella pagina delle statistiche utilizzano la libreria jqPlot. Per utilizzare jqPlot è stato sufficiente importare, all'interno del template `statistics.html`, tutti i file Javascript necessari, ognuno dedicato alla gestione di un diverso elemento, ed il foglio di stile CSS (Cascading Style Sheets) di jqPlot, tutti scaricabili dal sito ufficiale. Ad esempio lo script Javascript di jqPlot che gestisce i grafici a torta è denominato `jqplot.pieRenderer.js`, mentre lo script principale, contenente le funzioni di base, si chiama `jquery.jqplot.js`. Di seguito vediamo l'esempio del grafico a torta relativo alla distribuzione delle istanze per stato:

```
var countryNames = [{ country_names | tojson }];
var data = [];
for (var name in countryNames) {
    data.push([name, countryNames[name]]);
}
var options = {
    seriesDefaults: {
        renderer: $.jqplot.PieRenderer,
        rendererOptions: {
            showDataLabels: true,
        },
        shadow: false
    },
    legend: {
        show: true,
        location: 'e'
    },
    grid: {
        background: '#FFFFFF',
        borderWidth: 0,
        shadow: false
    }
};
$.jqplot(
    'country-piechart',
    [data],
    options
);
```

Vediamo che costruiamo prima la variabile `data` come una lista formata da liste di due elementi: il nome dello stato ed il numero di istanze in quello stato. Questi valori vengono passati, come accennato prima, dalla funzione corrispondente in `misc.py`, che si occupa di preparare tutti i parametri necessari ed invocare `render_template`. Successivamente si costruiscono le opzioni del grafico jqPlot: ad esempio la linea `renderer: $.jqplot.PieRenderer` indica che si vuole utilizzare un grafico a torta. Infine si invoca `$.jqplot()` indicandogli su quale elemento HTML costruire il grafico, con che dati e con quali opzioni. L'elemento su cui si inizializza il nuovo grafico jqPlot sarà quindi l'elemento HTML con id `country-piechart` ed avrà la seguente forma:

```
<div class="country-piechart-container">
  <div id="country-piechart"></div>
</div>
```

Il risultato di questa definizione è il grafico a torta visto in Figura 20.

Oltre a jqPlot la pagina delle statistiche fa uso anche di un'altra libreria, ovvero jVectorMap, che serve a visualizzare la mappa stilizzata in Figura 20 per la distribuzione delle istanze per stati. L'utilizzo di questa libreria è molto semplice: basta richiamare la funzione `vectorMap()` sull'elemento HTML corrispondente (come prima) e passargli come argomenti i dati, che saranno sotto forma di dizionario dove ad ogni nazione si associa il numero di istanze presenti, e le opzioni, come il colore degli stati, la grandezza e il colore dei marcatori, ecc.

Nella pagina della lista dei server viene invece utilizzata la libreria Javascript DataTables<sup>1</sup>, che permette di definire una tabella interattiva con possibilità di ricerca e ordinamento delle colonne.

Infine, all'interno della pagina dei dettagli di un'istanza, vengono utilizzate le API messe a disposizione da Google, per il servizio di Google Maps, per visualizzare la mappa interattiva relativa alla geolocalizzazione del server dell'istanza. Per maggiori informazioni a riguardo, si veda [14].

#### 6.2.6 Estrazione dei dati

Concludiamo questa parte dedicata all'implementazione di Cephalopod parlando del processo di estrazione di dati dalle istanze. Come già accennato, tutte le funzioni relative all'estrazione dei dati sono definite all'interno dello script `crawler.py`.

Come si è visto fin'ora, i modi per invocare le funzioni per l'estrazione dei dati sono molteplici: può essere fatto da uno scheduler Celery periodicamente,

<sup>1</sup> <https://datatables.net/>

oppure da linea di comando tramite lo script di gestione, o ancora tramite le azioni messe a disposizione dall'interfaccia dell'applicazione, le quali invieranno una richiesta ad un certo URL di Cephalopod il quale, tramite URL routing viene associato ad una funzione che si occupa di invocare la funzione di estrazione dati corrispondente.

Le funzioni definite in `crawler.py` sono quattro:

- `crawl()` : estrae i dati aggiuntivi da una determinata istanza;
- `geolocate()` : geolocalizza il server dell'istanza tramite il suo indirizzo IP;
- `crawl_instance()` : estrae i dati e geolocalizza il server per un'istanza (ovvero richiama `crawl()` e `geolocate()` );
- `crawl_all()` : invoca `crawl_instance()` su tutte le istanze attive.

Quindi il processo di estrazione di un'istanza, o di tutte le istanze, prevede in realtà l'estrazione dei dati aggiuntivi e la geolocalizzazione del server. La sola estrazione dei dati, come anche la geolocalizzazione, sono funzioni interne del crawler e non sono invocabili in alcun modo dall'esterno.

Molto semplicemente, l'operazione di estrazione dei dati aggiuntivi viene fatta innanzitutto formando l'URL della richiesta, per ogni endpoint specificato in `settings.cfg`, unendo l'URL di base dell'istanza e l'endpoint stesso. Quindi viene inviata una richiesta HTML a tale indirizzo, eventualmente passando il dizionario `header` che si può specificare sempre in `settings.cfg`. Dopodiché, il database verrà aggiornato, salvando la risposta alla richiesta nel campo `crawled_data` della classe `Instance` e salvando la data e ora attuale nel campo `crawl_date`. Quindi viene invocato `db.session.commit()` per rendere le modifiche effettive.

La fase di geolocalizzazione avviene invece utilizzando le API messe a disposizione dal servizio web gratuito FreeGeoIP<sup>1</sup> che permette di geolocalizzare un qualsiasi server su Internet tramite il proprio indirizzo IP. FreeGeoIP può fornire i dati di geolocalizzazione in tre formati: CSV (Comma Separated Values), XML (eXtensible Markup Language) e JSON. Dal momento che i dati di geolocalizzazione dovranno poi essere utilizzati in ambiente Javascript, il formato che ci serve in questo caso è JSON. L'URL a cui inviare la richiesta avrà quindi avere la seguente forma:

```
freegeoip.net/json/<instance_base_url>
```

Quindi, se ad esempio volessimo geolocalizzare il server della pagina di Google, dovremmo fare una richiesta all'URL `http://freegeoip.net/json/www.google.it`.

---

<sup>1</sup> <http://freegeoip.net/>



Una volta ottenuta la risposta in formato JSON contenente i dati di geolocalizzazione, questa potrà essere memorizzata nel database ed in particolare nel campo `geolocation` della classe `Instance` (si capisce, a questo proposito, come mai, quando abbiamo parlato del database, abbiamo detto che il campo di geolocalizzazione dev'essere di tipo JSON).

### 6.3 ADATTAMENTO DI INDICO

Parliamo adesso, in quest'ultima Sezione, di come Cephalopod è stato adattato a Indico, per permettere il corretto tracciamento di quest'ultimo.

Innanzitutto si sono dovute aggiungere, nel database di Indico, tutte le informazioni richieste da Cephalopod che non erano fino a prima presenti. Ad esempio, l'URL dell'istanza era ovviamente già presente all'interno delle impostazioni di Indico, mentre campi come l'UUID o lo stato del servizio di tracciamento (attivo/non attivo) non erano presenti e si sono dovuti aggiungere. Per quanto riguarda i campi aggiuntivi, non è stato necessario aggiungere alcun record all'interno del database, in quanto erano già ricavabili all'interno di Indico. I campi aggiuntivi scelti sono:

- numero di articoli, divisi per sezione;
- numero di eventi, divisi per sezione;
- versione di Indico;
- lingua;
- versione di Python;
- numero di file caricati totali;
- data di ultimo aggiornamento;
- numero di utenti.

Una volta assicuratici di avere a disposizione tutti i dati necessari per Cephalopod, si è creata una nuova pagina, all'interno del menu di Indico riservato agli amministratori, per gestire il sistema di tracciamento, ovvero per permettere all'istanza di interagire con Cephalopod. La pagina in questione è mostrata in Figura 22.

La voce "Status" indica se l'istanza è sincronizzata o meno con Cephalopod, ovvero se sono state fatte delle modifiche non ancora inviate a Cephalopod. Per implementare questo meccanismo viene fatta una richiesta HTML da Indico all'API di Cephalopod "Get instance": se l'istanza è registrata nel database di Cephalopod e i dati restituiti coincidono, allora l'istanza è sincronizzata, altrimenti no.

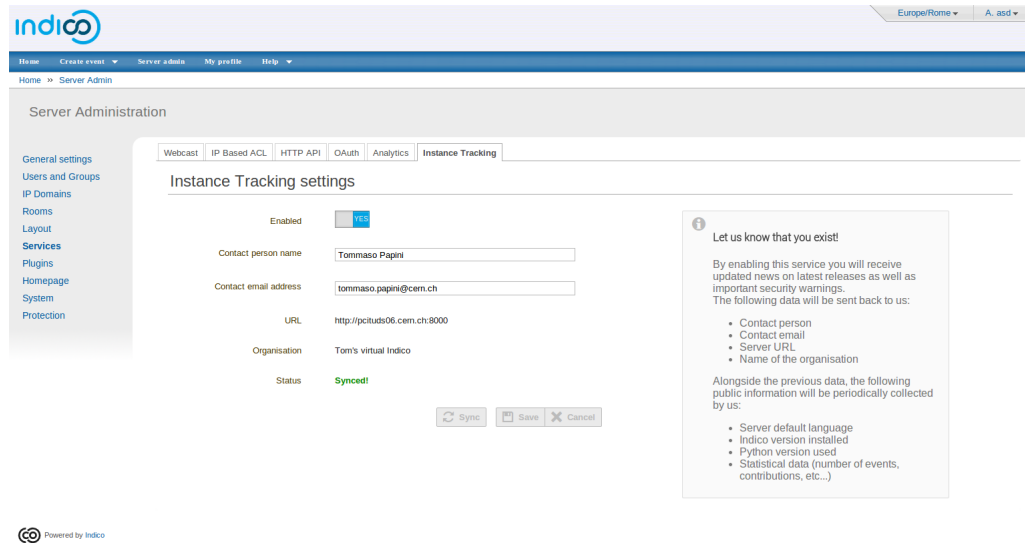


Figura 22: Pagina amministrativa per la gestione delle opzioni di tracciamento in Indico.

Il pulsante “Sync” serve invece ad avviare la sincronizzazione tra istanza e Cephalopod. In particolare, se l’istanza non è presente nel database di Cephalopod, questa viene creata tramite l’API “Create instance”, altrimenti questa viene semplicemente aggiornata con i nuovi dati tramite l’API “Update instance”.

Questa pagina serve sia come riepilogo dei dati principali scambiati con Cephalopod, sia per modificare e aggiornare gli stessi. Inoltre possiamo notare che accanto alla voce “Enabled” è presente un interruttore, che permette all’amministratore di attivare o disattivare in qualsiasi momento il tracciamento della propria istanza. Come già detto prima, questa è la scelta che è stata fatta dagli sviluppatori di Indico per il caso specifico di Indico: altri sviluppatori di un’altra applicazione potrebbero decidere di lasciare il tracciamento sempre abilitato di default, senza possibilità di disattivazione (anche se, per correttezza, sarebbe in ogni caso consigliabile avvertire gli utenti di tale scelta).

Inoltre è stato aggiunto anche un avviso popup, non intrusivo, sulla pagina principale del pannello di amministrazione di Indico, che avverte l’amministratore se l’istanza è fuori sincronia con Cephalopod. L’amministratore potrà scegliere se sincronizzare l’istanza subito oppure ignorare l’avviso, che scomparirà dopo poco da solo. In Figura 23 si può vedere come viene visualizzato tale messaggio.

### 6.3.1 Prima configurazione

L’ultima parte del processo di adattamento di Cephalopod a Indico riguarda la creazione di una pagina di prima configurazione per Indico.

Dal momento che si è scelto di avvertire l’amministratore di un’istanza, fin da

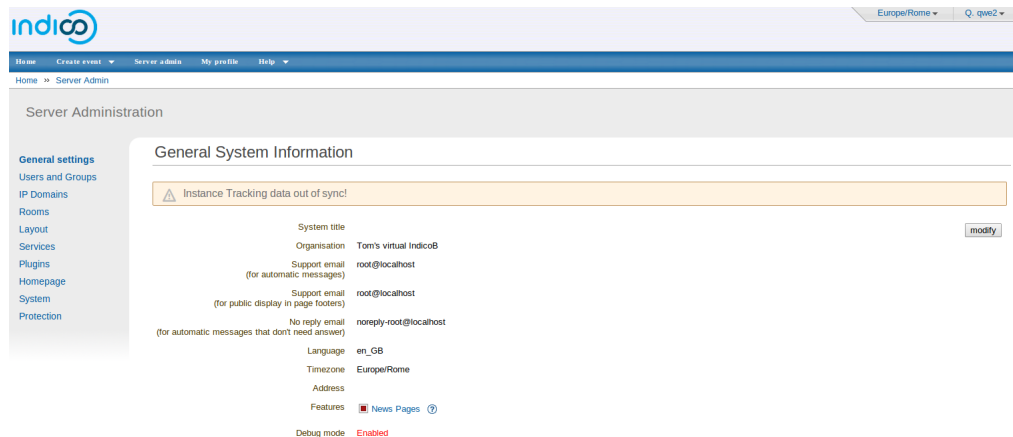



Figura 23: Avviso nella pagina principale di amministrazione di Indico che avverte della mancata sincronizzazione dell'istanza con Cephalopod.

subito, della possibilità di poter attivare il servizio di tracciamento (mettendolo in grado di non attivarlo nemmeno mai, se non lo volesse) è risultato necessario dover porre tale domanda all'amministratore non appena l'istanza di Indico è stata installata ed è pienamente funzionante.

L'idea era quella di mostrare una pagina dedicata, al posto della pagina principale di Indico, la prima volta in cui un amministratore effettuasse il primo accesso all'applicazione. Tuttavia creare una pagina apposita soltanto per una semplice domanda ("vuoi attivare il tracciamento? sì/no") ci è sembrato un po' uno spreco, al momento, così è stato deciso di cogliere l'occasione per creare una vera e propria pagina di prima configurazione per Indico, dove venissero chieste le informazioni di base dell'amministratore e dell'istanza e che presentasse uno stile moderno e rinnovato, in pieno accordo con la filosofia del Progetto KT.


Il risultato di questo lavoro è la pagina in Figura 24.

Come possiamo vedere, vengono richieste giusto alcune informazioni base e l'ultimo passo riguarda proprio il sistema di tracciamento, rendo chiaro fin da subito cosa esso comporti e facendo scegliere l'amministratore se partecipare o meno.



**Hello there!**  
 Indico needs just a little bit more data from you...

English




## Administrator Account

Insert here all basic information needed to create your account.

You can change any extra parameters (like your address or phone number) afterwards by accessing "Account Details" (from "My Profile").

First name  
 Family name  
 Email  
 Login  
 Password  
 Confirm password




## Site configuration

These are the default parameters for your Indico site.

Please note that this information will also be used for your own user profile. You can change it later.

Language English  
 Timezone Europe/Rome  
 Organization



## Let us know you exist!

Be part of the ever growing Indico community! Choose to receive updated news on latest releases (2-3 times per year) as well as important security warnings.

The following data will be sent back to us:

- Server URL
- Contact person
- Name of the organisation

Please note that **no private information** will ever be sent to the Indico Project and that you can disable it anytime (from "Server Admin", "Services", "Instance Tracking").

Count me in! NO  
 Contact name  
 Contact email

Finish

Figura 24: Nuova pagina di prima configurazione per amministratori di Indico.

## CONFERENCE CUSTOMIZATION PROTOTYPE

---

## ALTRI PROGETTI

---

### 8.1 I TICKET

### 8.2 NUOVA PAGINA DELLE STATISTICHE

### 8.3 INSTALLAZIONE SU WINDOWS SERVER

### Parte III

## NOTE E CONCLUSIONI

## CONCLUSIONI

---



## BIBLIOGRAFIA

---

- [1] CERN. Your career, 2015. URL <http://jobs.web.cern.ch/content/your-career>. (Citato a pagina 10.)
- [2] CERN. Culture and values, 2015. URL <http://jobs.web.cern.ch/content/culture-and-values>. (Citato a pagina 9.)
- [3] CERN. The structure of cern, 2015. URL <http://home.web.cern.ch/about/structure-cern>. (Citato a pagina 11.)
- [4] IT Dept. Indico service description. URL <http://information-technology.web.cern.ch/services/fe/indico>.
- [5] Pedro Ferreira. Improving the worldwide impact of Indico, Aprile 2013. URL <https://gist.github.com/pferreir/a819d1b99b9d225f4e0b>. (Citato a pagina 20.)
- [6] Pedro Ferreira. The road to indico 2.0. Aprile 2015. URL <https://indico.cern.ch/event/304944/session/6/contribution/61/material/slides/0.pdf>. (Citato a pagina 17.)
- [7] Brandon Fuller. The Beauty of CloudInit, Maggio 2011. URL <http://brandon.fuller.name/archives/2011/05/02/06.40.57/>.
- [8] Tommaso Papini. Cephalopod README.md, Giugno 2014. URL <https://github.com/oddlord/cephalopod/blob/master/README.md>.
- [9] Tommaso Papini. Indico KT Project: Improving the world-wide impact of Indico, Novembre 2014. URL <http://slides.com/oddlord/indico-kt-project>. (Citato a pagina 47.)
- [10] Sriram S. KVM and QEMU – do you know the connection?, Marzo 2014. URL <http://www.innervoice.in/blogs/2014/03/10/kvm-and-qemu/>.
- [11] CloudInit Team. Cloud-init Documentation, . URL <https://cloudinit.readthedocs.org/en/latest/index.html>. (Citato a pagina 22.)
- [12] Fabric Team. Fabric Documentation, . URL <http://docs.fabfile.org/en/1.10/>. (Citato a pagina 23.)
- [13] Github Team. Github Developer API, . URL <https://developer.github.com/v3/>. (Citato a pagina 44.)

- [14] Google Team. Google Maps JavaScript API, . URL <https://developers.google.com/maps/documentation/javascript/>. (Citato a pagina 69.)
- [15] Indico Team. Indico's user guide, . URL <https://indico.cern.ch/ihelp/html/UserGuide/index.html>.
- [16] Indico Team. ZODB, Settembre 2010. URL <http://old.indico-software.org/wiki/Dev/Technical/DB>. (Citato a pagina 17.)
- [17] Indico Team. Template engines, Febbraio 2011. URL <http://old.indico-software.org/wiki/Dev/Technical/TemplateEngines>. (Citato a pagina 17.)
- [18] Indico Team. WSGI, Febbraio 2011. URL <http://old.indico-software.org/wiki/Dev/Technical/WSGI>. (Citato a pagina 16.)
- [19] Indico Team. Flask, Luglio 2013. URL <http://old.indico-software.org/wiki/Dev/Technical/Flask/Usage>. (Citato a pagina 17.)
- [20] Indico Team. Indico features, 2014. URL <http://indico.github.io/features/>.
- [21] KVM Team. KVM Wiki, . URL [http://www.linux-kvm.org/page/Main\\_Page](http://www.linux-kvm.org/page/Main_Page). (Citato a pagina 24.)
- [22] Python Team. Creating Built Distributions, . URL <https://docs.python.org/2/distutils/builtdist.html>. (Citato a pagina 42.)
- [23] Python Team. Creating a Source Distribution, . URL <https://docs.python.org/2/distutils/sourcedist.html>. (Citato a pagina 42.)
- [24] QEMU Team. QEMU Wiki, . URL [http://wiki.qemu.org/Main\\_Page](http://wiki.qemu.org/Main_Page). (Citato a pagina 24.)
- [25] SQLAlchemy Team. Custom Types, . URL [http://docs.sqlalchemy.org/en/latest/core/custom\\_types.html#marshal-json-strings](http://docs.sqlalchemy.org/en/latest/core/custom_types.html#marshal-json-strings). (Citato a pagina 64.)
- [26] Geschreibsel von Christian Scholz. A small introduction to python eggs, Novembre 2007. URL <http://mrtopf.de/blog/en/a-small-introduction-to-python-eggs/>. (Citato a pagina 41.)
- [27] Wikipedia. CERN, Aprile 2015. URL <http://en.wikipedia.org/wiki/CERN>.
- [28] Wikipedia. CP violation, Aprile 2015. URL [http://en.wikipedia.org/wiki/CP\\_violation](http://en.wikipedia.org/wiki/CP_violation).

- [29] Wikipedia. NA48 experiment, Marzo 2015. URL [http://en.wikipedia.org/wiki/NA48\\_experiment](http://en.wikipedia.org/wiki/NA48_experiment).
- [30] Wikipedia. W and Z bosons, Aprile 2015. URL [http://en.wikipedia.org/wiki/W\\_and\\_Z\\_bosons](http://en.wikipedia.org/wiki/W_and_Z_bosons).
- [31] Wikipedia. Higgs boson, Maggio 2015. URL [http://en.wikipedia.org/wiki/Higgs\\_boson](http://en.wikipedia.org/wiki/Higgs_boson).
- [32] Wikipedia. Tar (computing): Software distribution, Marzo 2016. URL [https://en.wikipedia.org/wiki/Tar\\_\(computing\)#Software\\_distribution](https://en.wikipedia.org/wiki/Tar_(computing)#Software_distribution).  
(Citato a pagina 41.)