

**ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA**  
**SCUOLA DI INGEGNERIA E ARCHITETTURA: CAMPUS DI CESENA**

**Corso di laurea triennale in ingegneria biomedica**

**UTILIZZO DI GOOGLE KUBERNETES ENGINE PER  
UN'APPLICAZIONE IOT DI GESTIONE DELL'ACQUA  
IN AGRICOLTURA.**

**Relatore**

**Luca Roffia**

**Presentata da**

**Luca Leoni**

**Correlatore**

**Cristiano Aguzzi**

**Anno Accademico 2019/2020**

# INDICE

<b>INDICE</b> .....	<b>2</b>
<b>Indice delle figure</b> .....	<b>3</b>
<b>Abstract</b> .....	<b>4</b>
<b>KUBERNETES</b> .....	<b>5</b>
<b>Introduzione a Kubernetes</b> .....	<b>5</b>
<b>Glossario</b> .....	<b>6</b>
<b>Cluster di prova</b> .....	<b>9</b>
Introduzione .....	9
Minikube.....	10
Creazione dei deployment/pod .....	11
Creazione dei service .....	12
<b>Il comando kubectl</b> .....	<b>12</b>
<b>Google Kubernetes Engine (GKE)</b> .....	<b>17</b>
<b>Google Cloud Platform</b> .....	<b>17</b>
<b>Introduzione a Google Kubernetes Engine</b> .....	<b>18</b>
<b>Cluster di prova</b> .....	<b>19</b>
Creazione del cluster .....	19
Importazione del sistema di prova .....	20
GKE dashboard .....	21
Esposizione delle porte in rete .....	24
Volumi di memoria .....	25
Verifica corretto funzionamento dei volumi .....	27
<b>Costi</b> .....	<b>29</b>
<b>SWAMP su Google Kubernetes Engine</b> .....	<b>31</b>
<b>Introduzione a SWAMP</b> .....	<b>31</b>
<b>Sistema SWAMP</b> .....	<b>34</b>
<b>Modalità di accesso remoto a file condivisi</b> .....	<b>41</b>
<b>Ringraziamenti</b> .....	<b>42</b>
<b>Bibliografia e sitografia</b> .....	<b>43</b>

## Indice delle figure

FIGURA 1 CLUSTER E NODI IN KUBERNETES (FONTE [13])	6
FIGURA 2 RAPPRESENTAZIONE DI PODS IN KUBERNETES (FONTE [14])	7
FIGURA 3 SERVICE IN KUBERNETES (FONTE [15])	8
FIGURA 4 SCHEMA DELL'APPLICAZIONE DI PROVA	10
FIGURA 5 AVVIO DELLO STRUMENTO MINIKUBE	10
FIGURA 6 RISPOSTA AL COMANDO "KUBECTL GET NODES"	11
FIGURA 7 ESEMPIO DI FILE YAML NEL QUALE SONO PRESENTI LE INFORMAZIONI SU COME DEVE ESSERE CREATO IL DEPLOYMENT DEL DB VIRTUOSO	11
FIGURA 8 FILE YAML PER LA CREAZIONE DI UN SERVICE DI TIPO NODEPORT	12
FIGURA 9 COMANDO "KUBECTL GET PODS" E RELATIVA RISPOSTA	13
FIGURA 10 COMANDO "KUBECTL GET DEPLOYMENT" E RELATIVA RISPOSTA	13
FIGURA 11 COMANDO "KUBECTL GET SERVICE" E RELATIVA RISPOSTA	14
FIGURA 12 COMANDO "KUBECTL LOGS <NOME DEL POD>" E RELATIVA RISPOSTA	15
FIGURA 13 COMANDO "KUBECTL ATTACH <NOME DEL POD>" E RELATIVA RISPOSTA	15
FIGURA 14 COMANDO "MINIKUBE DASHBOARD"	15
FIGURA 15 VISIONE GENERALE DELLA DASHBOARD DI KUBERNETES	16
FIGURA 16 PERCORSO PER CREARE UN CLUSTER	19
FIGURA 17 BOTTONE DI ATTIVAZIONE DELLA CLOUD SHELL	20
FIGURA 18 SCHERMATA ALL'APERTURA DELLA CLOUD SHELL	20
FIGURA 20 UTILIZZO DEL COMANDO "KUBECTL ATTACH <NOME POD>"	21
FIGURA 21 FINESTRA DI INFORMAZIONI SUL PROGETTO NELLA DASHBOARD	22
FIGURA 22 FINESTRA API NELLA DASHBOARD	22
FIGURA 24 MENÙ DELLA SEZIONE KUBERNETES ENGINE	23
FIGURA 25 VISUALIZZAZIONE DEI DEPLOYMENT ATTRAVERSO LA VOCE "CARICHI DI LAVORO"	24
FIGURA 27 ACCESSO ALLA PAGINA DI AMMINISTRAZIONE DEL DB VIRTUOSO	25
FIGURA 28 ESEMPIO DI FILE YAML PER CREARE UN PERSISTENTVOLUMECLAIM	26
FIGURA 30 PAGINA DI CONFIGURAZIONE DEL SEPA	28
FIGURA 31 MODALITÀ DI INSERIMENTO DI NUOVI DATI	28
FIGURA 34 MAIL CONTENENTE LA FATTURA DEL NOSTRO ACCOUNT	30
FIGURA 35 ANDAMENTO GIORNALIERO DELLE SPESE	30
FIGURA 36 DESCRIZIONE DEL PROGETTO SWAMP (FONTE [26])	31
FIGURA 37 MAPPA CHE MOSTRA LE RICHIESTE DI IRRIGAZIONI PER I VARI CAMPI NELLA ZONA DEL PILOT ITALIANO DI REGGIO EMILIA	32
FIGURA 38 PAGINA WEB IN CUI SI VEDONO LE CONDIZIONI METEO E LA QUANTITÀ DI ACQUA NECESSARIA PER IRRIGARE QUESTO CAMPO	33
FIGURA 39 SCHEMA RIASSUNTIVO DELLE COMPONENTI DEL SISTEMA SWAMP	34
FIGURA 40 FILE YAML PER IL DEPLOYMENT DI SWAMP MAPPER (SINISTRA), FILE YAML PER IL DEPLOYMENT DI OBSERVATION LOGGER (DESTRA)	36
FIGURA 41 FILE YAML PER IL DEPLOYMENT DI MQTT AGENT (SINISTRA), FILE YAML PER IL DEPLOYMENT DI METER MAPPER (DESTRA)	36
FIGURA 42 FILE YAML PER IL DEPLOYMENT DI LEPIDA MAPPER (SINISTRA), FILE YAML PER IL DEPLOYMENT DI GUASPARI MAPPER (DESTRA)	37
FIGURA 43 FILE YAML PER IL DEPLOYMENT DI DEFAULT MAPPER (SINISTRA), FILE YAML PER IL DEPLOYMENT DI CRITERIA (DESTRA)	37
FIGURA 44 FILE YAML PER IL DEPLOYMENT DI CBEC ADAPTER (SINISTRA), FILE YAML PER IL DEPLOYMENT DI WDA (DESTRA)	38
FIGURA 45 FILE YAML PER IL SERVICE DI SWAMP MAPPER (SINISTRA), FILE YAML PER IL SERVICE DI OBSERVATION LOGGER (DESTRA)	38
FIGURA 46 FILE YAML PER IL SERVICE DI MQTT IOT AGENT (SINISTRA), FILE YAML PER IL SERVICE DI METER MAPPER (DESTRA)	39
FIGURA 48 FILE YAML PER IL SERVICE DI DEFAULT MAPPER (SINISTRA), FILE YAML PER IL SERVICE DI CRITERIA (DESTRA)	40

## Abstract

Questa tesi ha come obiettivo quello di studiare l'ambiente Kubernetes e di importare l'applicazione sviluppata per il progetto SWAMP su Google Kubernetes Engine (GKE). SWAMP (Smart Water Management Platform) è un progetto Europeo H2020 che ha come obiettivo quello di realizzare una piattaforma per la gestione dell'acqua in agricoltura, con lo scopo primario di ridurre al minimo il consumo. Essa si basa su IoT, cioè "Internet of Things", ed altre tecnologie avanzate come sensori e droni. Kubernetes è una piattaforma open source che permette di orchestrare tra loro applicazioni *containerizzate*, le quali hanno il vantaggio di essere facili da utilizzare e distribuire. Per creare i container contenenti le varie parti dell'applicazione complessiva si è utilizzata la tecnologia Docker.

Come primo approccio si è studiato Kubernetes su un sistema di prova, il quale contiene il database a grafo Virtuoso e il motore SEPA (alla base anche di SWAMP). In seguito si è testato il sistema di prova su Google Kubernetes Engine cercando di capire il suo funzionamento, con le relative similitudini a Kubernetes. Come ultimo passaggio è stata importata l'applicazione SWAMP su GKE. Per il corretto funzionamento dell'applicazione in GKE sono stati studiati anche una serie di concetti utili al corretto funzionamento dell'intera applicazione quali: volumi persistenti ed esposizione di servizi in rete.

# KUBERNETES



## Introduzione a Kubernetes

Kubernetes [2] è una piattaforma open-source in grado di gestire applicazioni in container, in particolar modo orchestrarle e renderle scalabili.

I container sono un ottimo modo per distribuire applicazioni, in quanto sono molto leggeri e hanno al loro interno tutto ciò che servirà all'applicazione per essere eseguita. Si consideri ora un sistema composto da diverse applicazioni containerizzate in comunicazione tra loro: qualora una dovesse interrompersi si bloccherebbe l'intero sistema. Inoltre se un'applicazione deve essere aggiornata, essa viene cancellata e poi rigenerata una volta eseguito l'upgrade. Kubernetes è lo strumento che permette di non interrompere mai un sistema che si basa su un insieme di applicazioni eseguite all'interno di container.

Per lo scopo di questa tesi Kubernetes servirà a orchestrare fra loro container che contengono ciascuno un'applicazione. Il grande vantaggio che dovrà fornire è quello di non interrompere mai il sistema, anche quando una delle applicazioni dovesse decadere oppure venisse aggiornata (caso di grande interesse).

Kubernetes si serve di una interfaccia che è lo strumento da riga di comando sul terminale: "kubectl".

## Glossario

Kubernetes ha una propria terminologia, che di seguito verrà specificata in quanto certi termini verranno utilizzati spesso nell'intera tesi.

- **Cluster:** Insieme di macchine, fisiche o virtuali, che eseguono le applicazioni e prendono il nome di “nodi”. In ogni cluster vi è almeno un “Worker Node”.
- **Nodo:** Può essere una macchina fisica o virtuale. Esso rappresenta un solo elemento nella molteplicità di nodi che possono creare il cluster. Il suo compito è quello di eseguire le applicazioni nei container e contiene i servizi di supporto richiesti. In ciascun cluster c'è sempre un nodo “master” che ha più poteri degli altri ed ha il compito di mantenere il corretto funzionamento del cluster.

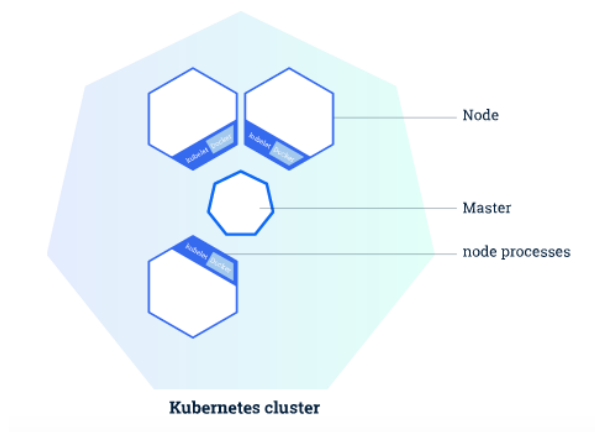


Figura 1 Cluster e nodi in Kubernetes (fonte [13])

- **Container:** Immagine leggera e facile di una applicazione da distribuire ed eseguire. Esso contiene tutte le informazioni per far eseguire correttamente l'applicazione. In questa tesi i container vengono creati attraverso la piattaforma “Docker” [5] e attraverso

l'interfaccia "docker hub" [6] si possono scaricare gratuitamente da internet.

- **Deployment:** Mezzo con il quale si può aggiornare lo stato presente scrivendo lo stato desiderato di un pod. Ciascun deployment fa riferimento a un solo pod del quale, però, si può scegliere di avere più replica set: in questo caso a ciascun deployment corrisponderanno più pod distinti, i quali contengono la stessa applicazione e funzionano esattamente allo stesso modo.
- **Pod:** Il più piccolo e semplice oggetto in Kubernetes rappresenta un gruppo di container nel cluster e, la singola istanza di un'applicazione. Nel pod è contenuta l'immagine Docker e tutte le modalità di esposizione dell'applicazione. In questa tesi a ogni pod corrisponde un solo container e quindi una sola applicazione da eseguire, in questo modo è risultato più semplice trattarli. Questa soluzione risulta comunque la più utilizzata in senso generale ed è denominata "one-container-per-pod". Ciascun pod nasce con un preciso IP che permette di individuarlo unicamente.

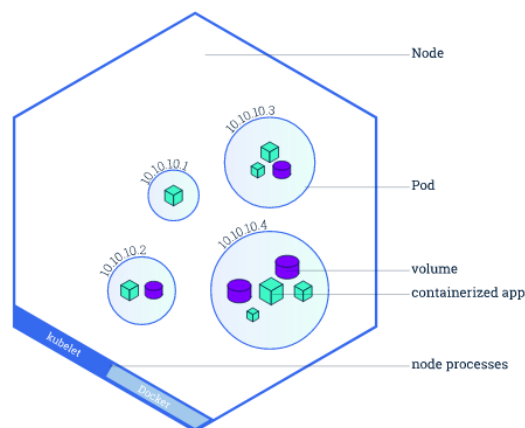


Figura 2 Rappresentazione di pods in Kubernetes (fonte [14])

- **Service:** Mezzo attraverso il quale si espongono i pod. L'esposizione può avvenire sia tramite la rete, accedendo dall'esterno, oppure tutta interna al cluster, quindi mettendo in comunicazione i pod tra loro. Esistono diverse tipologie di

service; in questa tesi si utilizza il service “NodePort”, che consente l’interfaccia tra i vari pod, ma anche l’esposizione su Internet.

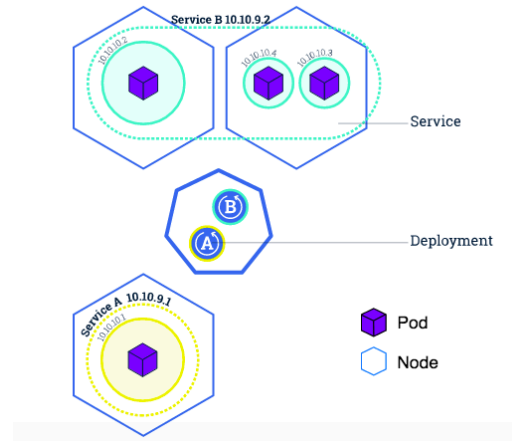


Figura 3 Service in Kubernetes (fonte [15])

- **Volume:** Unità di memoria utilizzata per salvare i vari dati generati e/o utilizzati dalle applicazioni. Ciascun pod contiene una memoria al proprio interno, che però va persa ogni volta che si rigenera il pod. L’idea è quella di creare uno spazio che mantenga i dati generati dalle applicazioni: quando un pod viene cancellato e ne viene generato immediatamente un altro, quest’ultimo prende i vari dati dal volume e permette all’intero cluster di non accorgersi della perdita di dati.
- **Dashboard:** Interfaccia web per l’utente. Si può utilizzare per avere una panoramica generale sul cluster, ma anche per eseguire degli aggiornamenti, creare nuovi pod e deployment, esporre servizi, verificare eventuali errori e riavviare i vari pod. In generale quasi tutte le operazioni si possono eseguire anche con codici di comando kubectl su terminale, ma utilizzare l’interfaccia dashboard agevola queste procedure.



## Cluster di prova

### Introduzione

Per iniziare si è creato un cluster di prova con i relativi deployment, pod e service per cercare di capire il funzionamento dell'ambiente Kubernetes. Il sistema sviluppato comprende 5 applicazioni implementate ciascuna con un pod e messe in comunicazione dai service di tipo NodePort.

Applicazioni:

- Database: Virtuoso [7]
- Engine: SEPA [8]
- Produttore
- Aggregatore
- Consumatore

Il database è l'elemento strutturalmente più in basso nel sistema, è alla base del funzionamento del SEPA. Quest'ultimo si posiziona al secondo posto nella struttura gerarchica sopra il database e sotto a produttore, aggregatore e consumatore. SEPA si basa su database per poter funzionare ed è il motore dell'intero sistema, in quanto acquisisce, elabora ed invia dati, per mezzo di un meccanismo di publish/subscribe.

Gli ultimi tre elementi costituiscono l'applicazione di prova. Produttore è un agente software che si occupa di contare i numeri in ordine crescente dallo zero e li invia all' engine. Aggregatore è l'applicazione che preleva i numeri dal SEPA, li somma e il risultato lo invia nuovamente ad esso. Infine il consumatore legge la somma.

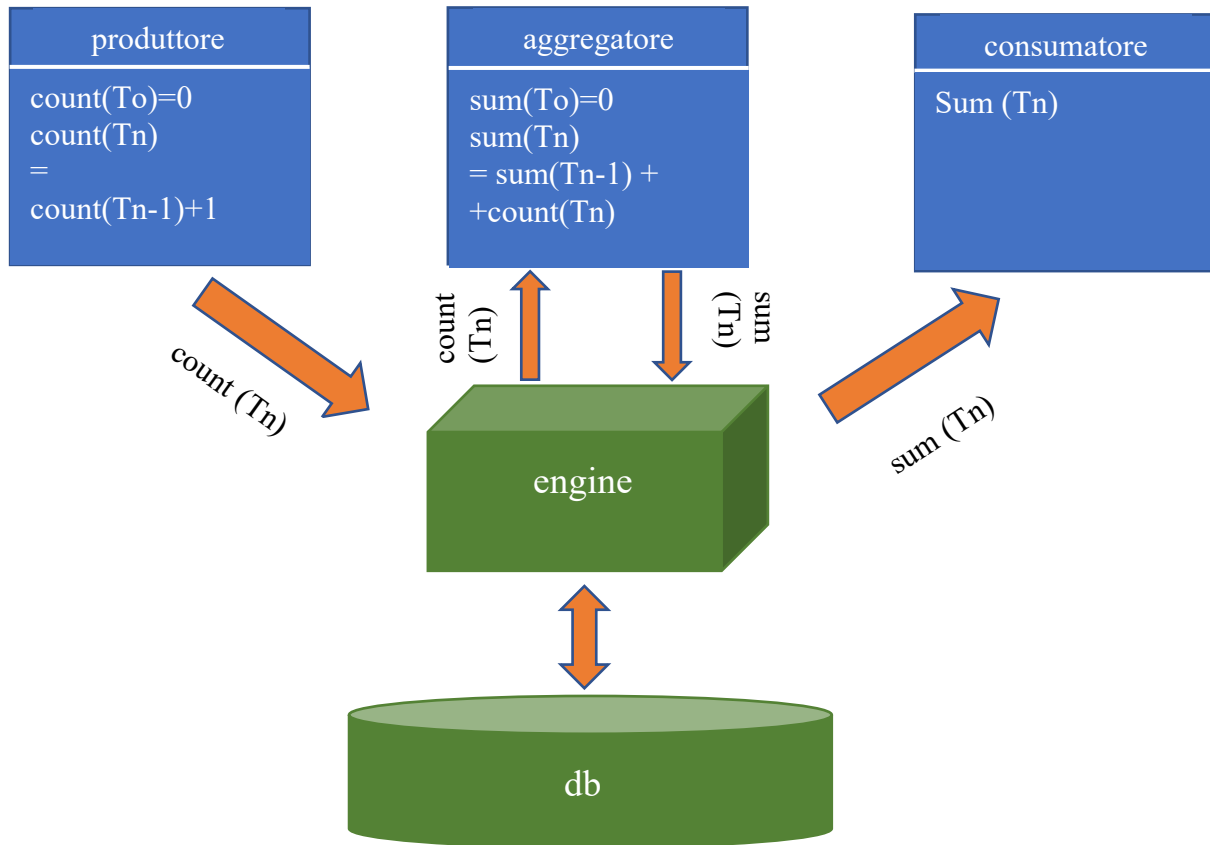


Figura 4 Schema dell'applicazione di prova

## Minikube

Minikube è lo strumento utilizzato in questa tesi per creare un cluster a nodo singolo su una macchina virtuale. Quindi dopo averlo scaricato dall'apposito sito [16] si lancia sul terminale con il comando “minikube start”.

```

MacBook-Pro-di-Luciano:~ lucianoleoni$ minikube start
🔥 minikube 1.11.0 is available! Download it: https://github.com/kubernetes/minikube/releases/tag/v1.11.0
💡 To disable this notice, run: 'minikube config set WantUpdateNotification false'

🤖 minikube v1.9.0 on Darwin 10.13.6
🌟 Using the hyperkit driver based on existing profile
🔄 Retarting existing hyperkit VM for "minikube" ...
🐳 Preparing Kubernetes v1.18.0 on Docker 19.03.8 ...
🌟 Enabling addons: dashboard, default-storageclass, storage-provisioner
🎉 Done! kubectl is now configured to use "minikube"

```

Figura 5 Avvio dello strumento Minikube

Come prima cosa si può verificare la creazione del nodo sul quale poi si lavorerà con il comando:

“*kubectl get nodes*”

Il risultato di questa ricerca è mostrato nella seguente immagine.

```

MacBook-Pro-di-Luciano:~ lucianoleoni$ kubectl get nodes
NAME        STATUS    ROLES    AGE   VERSION
minikube    Ready    master   67d   v1.18.0

```

Figura 6 Risposta al comando "kubectl get nodes"

## Creazione dei deployment/pod

Per la realizzazione dei deployment e di conseguenza dei relativi pod si possono percorrere diverse strade equivalenti:

- Riga di codice su terminale
- Documento JSON
- Documento YAML (opzione utilizzata in questa tesi)

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: db
  labels:
    app: db
spec:
  selector:
    matchLabels:
      app: db
  replicas: 1
  template:
    metadata:
      labels:
        app: db
    spec:
      containers:
      - env:
        - name: DEFAULT_GRAPH
          value: http://www.example.com/my-graph
        - name: SPARQL_UPDATE
          value: "true"
        image: tensorflow/virtuoso:1.3.1-virtuoso7.2.2
        name: db
        ports:
        - containerPort: 8890
          protocol: TCP

```

Figura 7 Esempio di file YAML nel quale sono presenti le informazioni su come deve essere creato il deployment del DB Virtuoso

Questo è il file che crea il deployment del database. Contiene tutte le informazioni necessarie al nostro scopo (si potrebbero specificare altre caratteristiche), da notare:

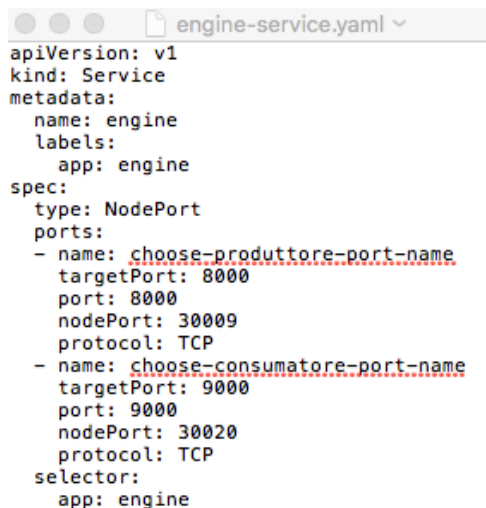
- “replicas:1” indica quanti pod differenti, ma tutti uguali a livello di funzionalità, questo deployment dovrà creare; è l’informazione che permette di scalare l’applicazione.
- “image: tensorflow/virtuoso:1.3.1-virtuoso7.2.2” indica l’immagine Docker che si utilizza per creare il container.

- “containerPort: 8890”: è la porta per accedere a questo container.

Infine, per eseguire il file ci sono due opzioni: tramite dashboard oppure tramite la riga di comando “*kubectl apply -f <nome file yml>.yaml*”.

## Creazione dei service

Ogni pod necessita di comunicare con alcuni degli altri, per questo vengono utilizzati i service. In questa tesi si utilizzerà quello di tipo: “NodePort”.



```
engine-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: engine
  labels:
    app: engine
spec:
  type: NodePort
  ports:
  - name: choose-produttore-port-name
    targetPort: 8000
    port: 8000
    nodePort: 30009
    protocol: TCP
  - name: choose-consumatore-port-name
    targetPort: 9000
    port: 9000
    nodePort: 30020
    protocol: TCP
  selector:
    app: engine
```

Figura 8 File YAML per la creazione di un service di tipo NodePort

Nella Figura 8 si vede il file yml utilizzato per creare il service del engine. Si vedono due porte perché una è quella per i dati in ingresso (publish) e l’altra è utilizzata per i dati in uscita (subscribe). Per ciascun “passaggio” di dati ci sono a sua volta due porte: “Port” indica la porta riconosciuta dal container interno al pod; ”NodePort” è la porta che si vede dall’esterno quando il servizio espone l’app.

## Il comando kubectl

In questo sotto-capitolo vengono riportati alcuni dei comandi da terminale kubectl spesso utilizzati in questa tesi.

- Visualizzazione dei pod: **“kubectl get pods”**.

Come si può notare nella seguente figura, la risposta a questo comando è la visualizzazione di tutti i pod presenti con i relativi nomi, replicaset, status: running corrisponde al corretto funzionamento; restarts indica quante volte è stato riavviato automaticamente il pod, ed infine l’età del pod.

```
MacBook-Pro-di-Luciano:~ lucianoleoni$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
aggregator-847479fcc9-mszwg         1/1    Running   1          6d17h
consumer-6bc8665fcf-ngnzd          0/1    CrashLoopBackOff   1          6d17h
db-6ffd67dcd7-64nwp                1/1    Running   1          6d17h
engine-6569c887d7-hhngn            1/1    Running   1          6d17h
produttore-847794f468-n72x2        1/1    Running   1          6d17h
MacBook-Pro-di-Luciano:~ lucianoleoni$
```

Figura 9 Comando "kubectl get pods" e relativa risposta

- Visualizzazione dei deployment: **“kubectl get deployment”**.

Questo comando serve a visualizzare tutti i deployment presenti. Anche in questo caso viene visualizzato il nome, mentre “ready”, il quale indica quante replica set nasceranno da ciascun deployment, segue il modello pronti/richiesti. “Up-to-date” indica il numero di repliche aggiornate per raggiungere lo stato desiderato, mentre “available” indica le repliche disponibili per gli utenti; infine viene visualizzata l’età dei deployment.

```
MacBook-Pro-di-Luciano:~ lucianoleoni$ kubectl get deployment
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
aggregator      1/1     1             1           6d18h
consumer        1/1     1             1           6d18h
db              1/1     1             1           6d18h
engine          1/1     1             1           6d18h
produttore      1/1     1             1           6d18h
MacBook-Pro-di-Luciano:~ lucianoleoni$
```

Figura 10 Comando "kubectl get deployment" e relativa risposta

- Visualizzazione dei service: **“kubectl get service”**.

Questo comando viene utilizzato per visionare tutti i service presenti, con le relative caratteristiche: nome, “type”, il quale indica la tipologia di service che, come già detto, in questo caso è di tipo “NodePort”. “Cluster-ip” indica l’ip unico del cluster su

cui è posizionato il pod relativo; mentre “external-ip”: visualizza l’ip che serve per accedere dall’esterno (in questo caso non è presente). “Port(s)” indica le porte utilizzate dal service per esporre il pod, (<porta interna dell’applicazione>:<nodeport del service>; fondamentali entrambe altrimenti l’informazione non riesce ad avere il giusto passaggio per accedere al pod) e TCP indica il tipo di protocollo, infine anche in questo caso l’età del service.

```
MacBook-Pro-di-Luciano:~ lucianoleoni$ kubectl get service
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
db            NodePort     10.101.233.53 <none>         8890:30012/TCP   74d
engine       NodePort     10.98.74.102  <none>         8000:30009/TCP,9000:30020/TCP 74d
kubernetes   ClusterIP    10.96.0.1     <none>         443/TCP          74d
produttore   NodePort     10.99.133.127 <none>         8000:30010/TCP   74d
MacBook-Pro-di-Luciano:~ lucianoleoni$
```

Figura 11 Comando "kubectl get service" e relativa risposta

- Visualizzazione dell’esecuzione dell’applicazione:

“**kubectl logs <nome del pod>**”.

Con questo comando è possibile visualizzare sul terminale il corretto funzionamento dell’applicazione. Nella seguente figura è riportata l’esecuzione del produttore. E’ importante notare che viene visualizzato il contenuto eseguito fino al momento in cui è lanciato il comando. Nel caso in cui le applicazioni siano dinamiche e quindi, come in questo caso, un nuovo numero venga prodotto in ogni secondo, non verranno visualizzati i numeri prodotti temporalmente dopo l’esecuzione del comando sopra riportato.

```
MacBook-Pro-di-Luciano:~ lucianoileoni$ kubectl logs produttore-847794f468-kbtsz
2020-07-04T10:37:42,432 [INFO ] main (App.java:29) Create producer
2020-07-04T10:37:42,875 [INFO ] main (App.java:32) Set UUID
2020-07-04T10:37:42,878 [INFO ] Thread-1 (App.java:42) docker:Producer_b673291a_502f_4fac_a5db_c5901c98688d update counter 0
2020-07-04T10:37:43,334 [timing] Thread-1 (Timings.java:19) 1593859063333,417035915,ENDPOINT_UPDATE_TIME
2020-07-04T10:37:44,345 [INFO ] Thread-1 (App.java:42) docker:Producer_b673291a_502f_4fac_a5db_c5901c98688d update counter 1
2020-07-04T10:37:44,389 [timing] Thread-1 (Timings.java:19) 1593859064388,41086435,ENDPOINT_UPDATE_TIME
2020-07-04T10:37:45,391 [INFO ] Thread-1 (App.java:42) docker:Producer_b673291a_502f_4fac_a5db_c5901c98688d update counter 2
2020-07-04T10:37:45,407 [timing] Thread-1 (Timings.java:19) 1593859065406,14050322,ENDPOINT_UPDATE_TIME
2020-07-04T10:37:46,408 [INFO ] Thread-1 (App.java:42) docker:Producer_b673291a_502f_4fac_a5db_c5901c98688d update counter 3
2020-07-04T10:37:46,433 [timing] Thread-1 (Timings.java:19) 1593859066433,22549267,ENDPOINT_UPDATE_TIME
2020-07-04T10:37:47,435 [INFO ] Thread-1 (App.java:42) docker:Producer_b673291a_502f_4fac_a5db_c5901c98688d update counter 4
2020-07-04T10:37:47,454 [timing] Thread-1 (Timings.java:19) 1593859067454,16026754,ENDPOINT_UPDATE_TIME
2020-07-04T10:37:48,455 [INFO ] Thread-1 (App.java:42) docker:Producer_b673291a_502f_4fac_a5db_c5901c98688d update counter 5
2020-07-04T10:37:48,475 [timing] Thread-1 (Timings.java:19) 1593859068475,18150478,ENDPOINT_UPDATE_TIME
2020-07-04T10:37:49,476 [INFO ] Thread-1 (App.java:42) docker:Producer_b673291a_502f_4fac_a5db_c5901c98688d update counter 6
2020-07-04T10:37:49,497 [timing] Thread-1 (Timings.java:19) 1593859069497,18650628,ENDPOINT_UPDATE_TIME
2020-07-04T10:37:50,501 [INFO ] Thread-1 (App.java:42) docker:Producer_b673291a_502f_4fac_a5db_c5901c98688d update counter 7
2020-07-04T10:37:50,517 [timing] Thread-1 (Timings.java:19) 1593859070517,14874153,ENDPOINT_UPDATE_TIME
```

Figura 12 Comando "kubectl logs <nome del pod>" e relativa risposta

Un'altra opzione per vedere dinamicamente l'esecuzione dell'applicazione è il comando **"kubectl attach <nome del pod>"**. In questo caso, fino al momento in cui sul terminale non venga stoppato il comando, si vedranno comparire sequenzialmente i numeri prodotto dal pod.

```
MacBook-Pro-di-Luciano:~ lucianoileoni$ kubectl attach produttore-847794f468-kbtsz
Defaulting container name to produttore.
Use 'kubectl describe pod/produttore-847794f468-kbtsz -n default' to see all of the containers in this pod.
If you don't see a command prompt, try pressing enter.
2020-07-04T12:51:27,654 [INFO ] Thread-1 (App.java:42) docker:Producer_b673291a_502f_4fac_a5db_c5901c98688d update counter 2552
2020-07-04T12:51:27,679 [timing] Thread-1 (Timings.java:19) 1593867087679,22377589,ENDPOINT_UPDATE_TIME
2020-07-04T12:51:28,681 [INFO ] Thread-1 (App.java:42) docker:Producer_b673291a_502f_4fac_a5db_c5901c98688d update counter 2553
2020-07-04T12:51:28,704 [timing] Thread-1 (Timings.java:19) 1593867088704,20031828,ENDPOINT_UPDATE_TIME
2020-07-04T12:51:29,707 [INFO ] Thread-1 (App.java:42) docker:Producer_b673291a_502f_4fac_a5db_c5901c98688d update counter 2554
2020-07-04T12:51:29,748 [timing] Thread-1 (Timings.java:19) 1593867089748,35417792,ENDPOINT_UPDATE_TIME
2020-07-04T12:51:30,750 [INFO ] Thread-1 (App.java:42) docker:Producer_b673291a_502f_4fac_a5db_c5901c98688d update counter 2555
2020-07-04T12:51:30,777 [timing] Thread-1 (Timings.java:19) 1593867090776,18554958,ENDPOINT_UPDATE_TIME
^C
MacBook-Pro-di-Luciano:~ lucianoileoni$
```

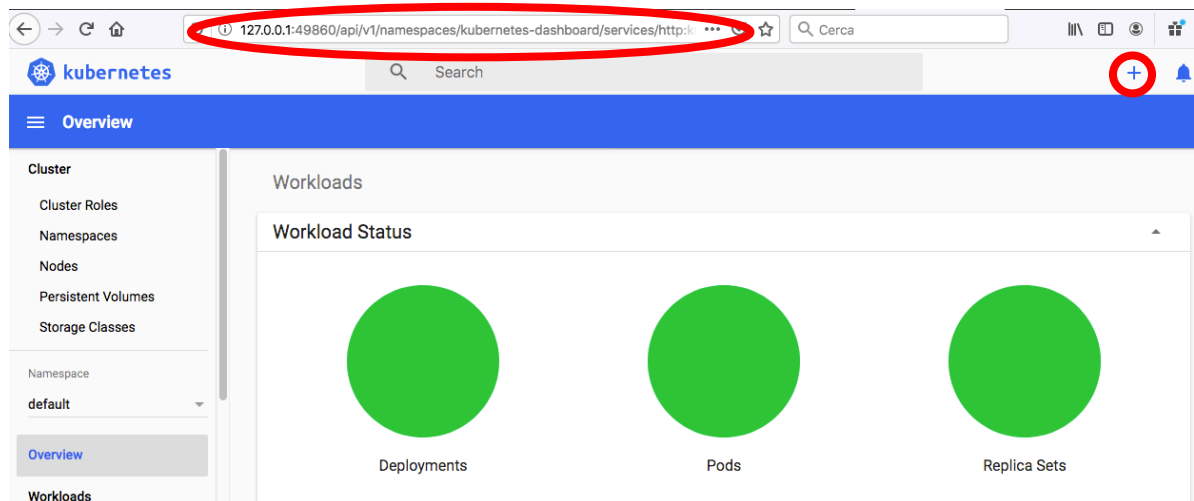
Figura 13 Comando "kubectl attach <nome del pod>" e relativa risposta

- Apertura automatica sul browser della dashboard: **"minikube dashboard"**.

Questo comando permette di creare il link e quindi di aprire la dashboard del cluster. Il terminale sul quale si esegue il comando dovrà essere dedicato solamente a questa operazione, in caso contrario la dashboard smetterà di funzionare.

```
MacBook-Pro-di-Luciano:~ lucianoileoni$ minikube dashboard
🔍 Verifying dashboard health ...
🚀 Launching proxy ...
🔍 Verifying proxy health ...
🔗 Opening http://127.0.0.1:49860/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
```

Figura 14 Comando "minikube dashboard"



*Figura 15 Visione generale della dashboard di Kubernetes*

Notare in Figura 15 l'indirizzo web utilizzato per aprire la dashboard ed il bottone “+” per aggiungere file yaml, e caricarli direttamente dal computer.



## Google Kubernetes Engine (GKE)



### Google Cloud Platform

Google Cloud Platform è la piattaforma lanciata da Google nel 2011. Fornisce servizi su vasta scala tra cui: machine learning, intelligenza artificiale, big data, gestione e distribuzione di applicazioni. La piattaforma è a pagamento, eccetto una prova gratuita iniziale usufruibile da tutti gli utenti che si avvicinano ad essa.

Tra i servizi principali offerti ci sono Google App Engine, Google Compute Engine, Google Kubernetes Engine.

Google App Engine consente agli utenti di distribuire il codice delle loro applicazioni senza dover più pensare alla fase di distribuzione, in quanto è gestita dalla piattaforma che crea più istanze se l'applicazione ha grande richiesta, lasciando al programmatore il solo impegno nello sviluppo del codice.

Google Compute Engine si occupa di mettere a disposizione macchine virtuali altamente personalizzabili dagli utenti. Questa sezione della piattaforma consente l'utilizzo di container per la distribuzione di applicazioni. Google Kubernetes Engine è la sezione in cui si utilizza il software Kubernetes (di proprietà Google) per la gestione, distribuzione ed orchestrazione di applicazioni. In questa sezione, che è quella utilizzata in questa tesi, molti aspetti, come i codici da utilizzare, l'impostazione dei cluster, le modalità in cui vengono gestiti i container, sono identici a quelli trattati nel capitolo precedente.

## Introduzione a Google Kubernetes Engine

Google Kubernetes Engine è quella sezione della Google Cloud Platform finalizzata all'utilizzo di Kubernetes per la gestione, orchestrazione e distribuzione di applicazioni containerizzate. Google porta diversi vantaggi rispetto all'utilizzo di Kubernetes in locale:

- **Identità di accesso:** ogni utente può accedere da qualsiasi computer attraverso il login e sarà abilitato al suo ruolo.
- **Opzioni cluster:** nel momento in cui si crea un nuovo cluster lo si può sviluppare conforme alle proprie esigenze scegliendo il luogo geografico dove crearlo, quantità di memoria, quantità di CPU, quantità di RAM da dedicare al cluster.
- **Scalabilità automatica:** verso l'alto o verso il basso in base alle richieste.
- **Upgrade automatico:** permette di mantenere il cluster sempre aggiornato alla nuova release di Kubernetes, ovviamente fino a che il cluster supporta le nuove versioni.
- **Limite di risorse:** Kubernetes permette di dedicare una quantità specifica di CPU e di memoria (RAM) per ogni pod, così da organizzare al meglio i carichi di lavoro nel cluster.
- **Monitoraggio costante delle attività visibile sulla dashboard.**
- **Fatturazione al secondo:** si paga il servizio in base al tempo di utilizzo al secondo.

## Cluster di prova

Dopo aver testato il corretto funzionamento del cluster di prova su Kubernetes localmente, si è importato il medesimo cluster su Google Kubernetes Engine.

Come primo passaggio ci si deve registrare su Google Cloud Platform con un account che poi permetterà di accedere e ritrovare sempre il progetto nella situazione in cui lo si era lasciato, anche a distanza di diverso tempo (mesi attestato). Allo stesso account è inviata la mail di fatturazione.

## Creazione del cluster

Dopo aver effettuato l'accesso sulla piattaforma ci si trova in una schermata in cui si deve cliccare il pulsante “menù di navigazione” > “kubernetes engine” > “cluster”.

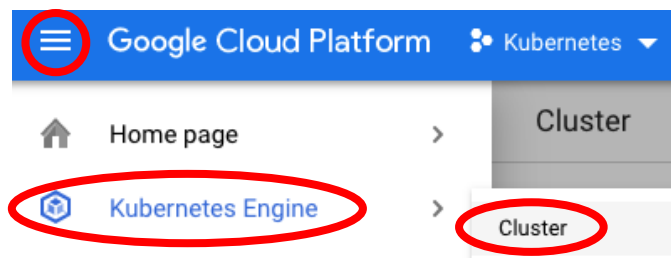


Figura 16 Percorso per creare un cluster

A questo punto si clicca il bottone “crea Cluster”. Nella nuova schermata bisogna selezionare alcune caratteristiche del cluster (di seguito sono riportate le caratteristiche consigliate da Google per creare un cluster poco costoso, utilizzate anche in questa prova):

- Nome del cluster: my-first-cluster-1
- Zona del cluster: us-central1-c
- Versione: canale di rilascio rapido invece della versione predefinita

- Tipo di macchina: g1-small invece di n1-standard-1
- Dimensioni del disco di avvio: 32 GB invece di 100 GB
- Scalabilità automatica: disabilitata
- Kubernetes Engine Monitoring: disabilitato

Infine si clicchi il pulsante “crea cluster” .

Ora accedendo sulla sezione “Kubernetes engine” > “cluster” ci si ritrova il cluster appena creato e selezionandolo si ritrovano le caratteristiche scelte nel momento della creazione. Si deve considerare che alcune di esse potranno essere modificate in seguito, mentre altre rimarranno invariate, come per esempio la zona del cluster. Da questa pagina si può anche scegliere di eliminare il cluster cancellandolo dai server e bloccando la relativa fatturazione (bisogna tenere in considerazione che mentre ci sono carichi di lavoro attivi verrà addebitato un costo all’utente, quindi si consiglia sempre di eliminare tutto quello che non si utilizza più).

## Importazione del sistema di prova

A questo punto occorre aprire la cloud shell di Google con un bottone sulla schermata in alto a destra.

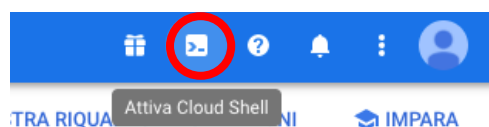


Figura 17 Bottone di attivazione della cloud shell



Figura 18 Schermata all'apertura della cloud shell

Nella cloud shell i comandi utilizzati nel precedente capitolo su Kubernetes rimangono validi quindi, per esempio, se si digitasse il comando “kubectl get pods” il risultato sarebbe la visualizzazione dei pod presenti nel progetto.

```
luca_leoni@cloudshell:~ (kubernetes-276210)$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
aggregatore-8649bcf46d-mcgd8        1/1     Running   0           41d
db-777cf6949b-5gjjvh                1/1     Running   0           40d
engine-6dfc7d6488-47f96            1/1     Running   0           44d
produttore-799cfb84cf-xc6nx         1/1     Running   0           5d1h
```

Figura 19 Esecuzione del comando "kubectl get pods"

Per importare o scrivere i file yaml ci sono due possibilità: scriverli direttamente tramite l’editor oppure importarli dal computer attraverso “espandi” > “carica file” (in Figura 18 sono cerchiare le due possibilità).

I file yaml utilizzati in questo approccio sono gli stessi utilizzati nel capitolo 1, quindi non saranno riportati nuovamente.

Anche in questo caso vengono creati i cinque deployment, con i relativi pod, e i cinque service che permettono di far comunicare tra loro i pod. Caricato il sistema di prova per verificare il corretto funzionamento si consiglia di lanciare il consumatore, ultimo anello nella catena, che se funzionante garantisce il corretto funzionamento dell’intero sistema. Il comando da utilizzare è: ”kubectl attach <nome pod>”.

Nella schermata si vede il consumatore che ogni volta che riceve il dato aggiornato dal SEPA lo scrive, come riportato in Figura 20.

```
luca_leoni@cloudshell:~ (kubernetes-276210)$ kubectl attach consumer-84fb648c8-wfkjb
Defaulting container name to consumer.
Use 'kubectl describe pod/consumer-84fb648c8-wfkjb -n default' to see all of the containers in this pod.
If you don't see a command prompt, try pressing enter.
2020-06-29T13:37:25,532 [DEBUG] Grizzly(2) (Consumer.java:124) onSemanticEvent: sepa://subscription/8523c3e1-3053-4ec7-b7b2-168c87bded18 258
2020-06-29T13:37:25,534 [INFO ] Grizzly(2) (App.java:41) {"head":{"vars":{"subject","sum"},"results":{"bindings":[{"subject":{"type":"url","value":"http://sepa/d
ocker#Producer_413fc9f3_b0e5_4fd5_bb62_6d56ec35d373"},"sum":{"type":"literal","value":"5869299029677"}}]}}
2020-06-29T13:37:26,542 [DEBUG] Grizzly(1) (Consumer.java:124) onSemanticEvent: sepa://subscription/8523c3e1-3053-4ec7-b7b2-168c87bded18 259
2020-06-29T13:37:26,542 [INFO ] Grizzly(1) (App.java:41) {"head":{"vars":{"subject","sum"},"results":{"bindings":[{"subject":{"type":"url","value":"http://sepa/d
ocker#Producer_413fc9f3_b0e5_4fd5_bb62_6d56ec35d373"},"sum":{"type":"literal","value":"5869299464507"}}]}}
```

Figura 20 Utilizzo del comando "kubectl attach <nome pod>"

## GKE dashboard

La dashboard della piattaforma Google Cloud è molto più completa di quella utilizzata da Kubernetes.

La schermata iniziale, alla quale si ritorna da qualunque posizione cliccando “Google Cloud Platform”, mostra le caratteristiche di base del progetto, quali:

- Informazioni sul progetto (nome, id, numero)

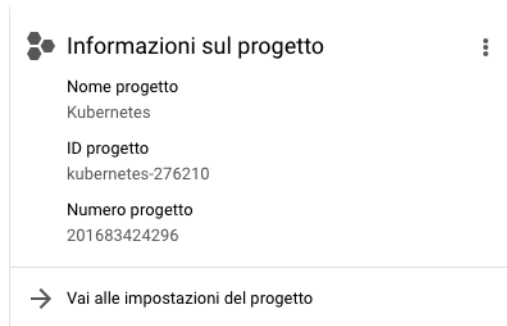


Figura 21 Finestra di informazioni sul progetto nella dashboard

- Risorse utilizzate
- Stato di Google Cloud Platform
- Fatturazione
- API: grafico richieste/secondo.

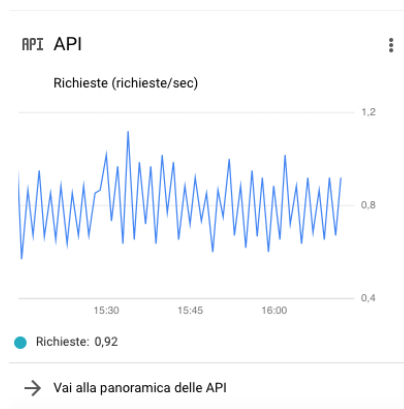


Figura 22 Finestra API nella dashboard

- Compute engine: CPU%, cioè monitora costantemente l'utilizzo di CPU.

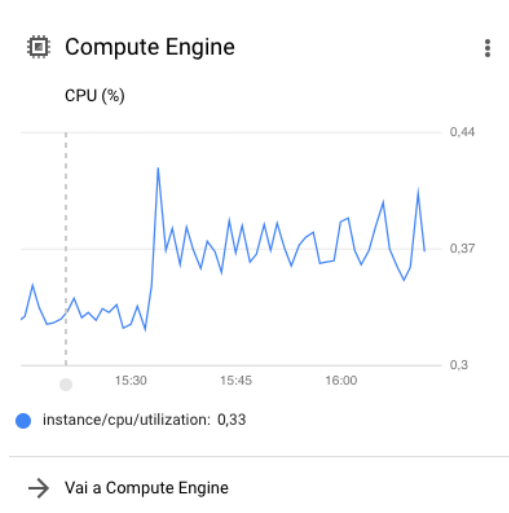


Figura 23 Finestra Compute Engine nella dashboard

Per entrare nella sezione specifica del progetto in esame si deve accedere alla sezione riservata a Kubernetes Engine cliccando i bottoni: “menù di navigazione” > “Kubernetes Engine”.

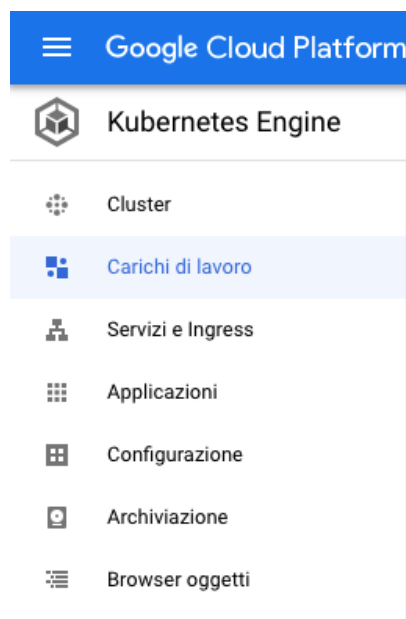


Figura 24 Menù della sezione Kubernetes Engine

Le voci del menù utilizzate in questa tesi sono: Cluster, Carichi di lavoro, servizi e Ingress, Archiviazione.

Accedendo relativamente a carichi di lavoro e servizi si trovano i deployment ed i service creati fino a quel momento (esempio Figura 25).

<input type="checkbox"/>	Nome ↑	Stato	Tipo	Pod	Spazio dei nomi	Cluster	Pod in esecuzione
<input type="checkbox"/>	aggregator	✔ OK	Deployment	1/1	default	cluster-prova	1
<input type="checkbox"/>	consumer	✔ OK	Deployment	1/1	default	cluster-prova	1
<input type="checkbox"/>	db	✔ OK	Deployment	1/1	default	cluster-prova	1
<input type="checkbox"/>	engine	✔ OK	Deployment	1/1	default	cluster-prova	1
<input type="checkbox"/>	produttore	✔ OK	Deployment	1/1	default	cluster-prova	1

Figura 25 Visualizzazione dei deployment attraverso la voce "carichi di lavoro"

## Esposizione delle porte in rete

Come già descritto in precedenza si è utilizzato il service di tipo “NodePort”, in quanto assicura ad ogni pod una porta ben definita ed unica poiché la si può scegliere personalmente. Inoltre, questo tipo di service garantisce una esposizione sia dentro il cluster, quindi tra i pod stessi, sia sulla rete. Questo consente di poter accedere anche dall’esterno del cluster all’applicazione. Per potervi accedere bisogna conoscere l’indirizzo IP e la porta da digitare nella barra degli indirizzi sul browser.

Esiste una procedura per poter esporre il servizio in rete ed è la seguente:

- digitare sul terminale il comando: ”kubectl get nodes --output wide” la cui relativa risposta è visualizzare le caratteristiche dei nodi, tra cui l’IP esterno di interesse per questo scopo.

```
luca_leoni@cloudshell:~ (kubernetes-276210) $ kubectl get nodes --output wide
NAME                                STATUS    ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE                                     KERNEL-
VERSION   CONTAINER-RUNTIME
gke-cluster-prova-default-pool-35478408-8jvz  Ready    <none>   45d   v1.14.10-gke.36   10.172.0.4    34.65.95.142   Container-Optimized OS from Google         4.14.13
8+       docker://18.9.7
gke-cluster-prova-default-pool-35478408-fx4x  Ready    <none>   45d   v1.14.10-gke.36   10.172.0.2    34.65.23.15    Container-Optimized OS from Google         4.14.13
8+       docker://18.9.7
gke-cluster-prova-default-pool-35478408-gr0h  Ready    <none>   45d   v1.14.10-gke.36   10.172.0.3    34.65.136.66   Container-Optimized OS from Google         4.14.13
8+       docker://18.9.7
```

Figura 26 Risposta al comando "kubectl get nodes -output wide"

- digitare il seguente comando:  
“gcloud compute firewall-rules create <nome del permesso>  
--allow tcp:<porta del nodo>” .



Questo comando crea le firewall-rules per permettere un traffico TCP sulla porta indicata.

Per verificare la corretta esposizione occorre scrivere sulla barra degli indirizzi del browser: “external ip:nodeport”. Per esempio in questo caso verifico l’esposizione del db virtuoso, il cui external IP è: 34.65.136.66; mentre la porta è: 30012. Quindi digitando sul browser: “34.65.136.66:30012” si vedrà aprire la pagina di Virtuoso, come rappresentato in Figura 27.



Figura 27 Accesso alla pagina di amministrazione del DB Virtuoso

## **Volumi di memoria**

Il volume di memoria è una sezione che permette di salvare dei dati, che non vanno persi qualora un pod venisse rigenerato.

In questo caso il database è l’applicazione che deve mantenere i dati altrimenti il SEPA ogni volta che il pod db dovesse decadere perderebbe tutte le informazioni passate.

In questo caso come primo passaggio si deve creare un volume. In particolar modo bisogna creare un “PersistentVolumeClaim”, cioè una richiesta di volume permanente, nella quale si specifica la modalità di utilizzo e la quantità di spazio da dedicare. In Figura 28 si può vedere il file yaml utilizzato per creare un “PersistentVolumeClaim”.

```

persistentvolumeclaim.yaml x
1  kind: PersistentVolumeClaim
2  apiVersion: v1
3  metadata:
4    name: podpvc
5  spec:
6    accessModes:
7    - ReadWriteOnce
8    storageClassName: singlewriter-standard
9    resources:
10   requests:
11   storage: 30Gi

```

Figura 28 Esempio di file YAML per creare un PersistentVolumeClaim

Da notare nel file in Figura 28 le voci:

- “accessMode” nella quale va specificata la modalità di utilizzo. In questo caso si utilizza “ReadWriteOnce”, cioè il volume è utilizzato in lettura e scrittura da un solo nodo. In alternativa si può utilizzare “ReadWriteMany”, cioè il volume è utilizzabile in lettura e scrittura da più nodi. Infine “ReadOnlyMany” indica l’utilizzo del volume in sola lettura da più nodi.
- “storage” nel quale va indicato la quantità di memoria da dedicare in questo volume.

Una volta scritto il file yaml come sempre per applicarlo si digita sul terminale: “kubectl apply -f persistentvolumeclaim.yaml”.

Una volta creato lo si dovrebbe trovare anche nella voce del menù “Archiviazione” (visibile anche in Figura 24).

A questo punto bisogna modificare il deployment del db per fare in modo che si specifichi il salvataggio di dati su questo volume persistente. In Figura 29 si vede il nuovo file yaml del db utilizzato per salvare i dati sul volume permanente.

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: db
5    labels:
6      app: db
7  spec:
8    selector:
9      matchLabels:
10     app: db
11    replicas: 1
12    template:
13      metadata:
14        labels:
15          app: db
16      spec:
17        containers:
18          - env:
19            - name: DEFAULT_GRAPH
20              value: http://www.example.com/my-graph
21            - name: SPARQL_UPDATE
22              value: "true"
23            image: tenforce/virtuoso:1.3.1-virtuoso7.2.2
24            name: db
25            volumeMounts:
26              - mountPath: /data
27
28              name: mypvc
29            ports:
30              - containerPort: 8890
31                protocol: TCP
32            volumes:
33              - name: mypvc
34                persistentVolumeClaim:
35                  claimName: podpvc
36                  readOnly: false

```

Figura 29 Esempio di file YAML per creare un pod che utilizzi persistent volume claim precedentemente creata per salvare i dati generati dall'applicazione

In questo file si può vedere che rispetto alla Figura 7 sono state aggiunte le voci: “volumeMounts” che specifica il percorso da seguire per inviare i dati e “volumes” che specifica il volume da utilizzare.

## Verifica corretto funzionamento dei volumi

Per verificare il corretto funzionamento del sistema si deve in primo luogo accedere al SEPA e inserire dei dati. Si deve cercare sul browser l’indirizzo <http://mml.arces.unibo.it/apps/dashboard/>, cioè la dashboard del SEPA. Nella prima schermata che si vede in Figura 30, viene caricato un file jsap nel quale sono contenute le informazioni di configurazione; in particolare il file contiene l’indirizzo IP e le porte alle quali deve accedere, che sono le stesse esposte come spiegato precedentemente.

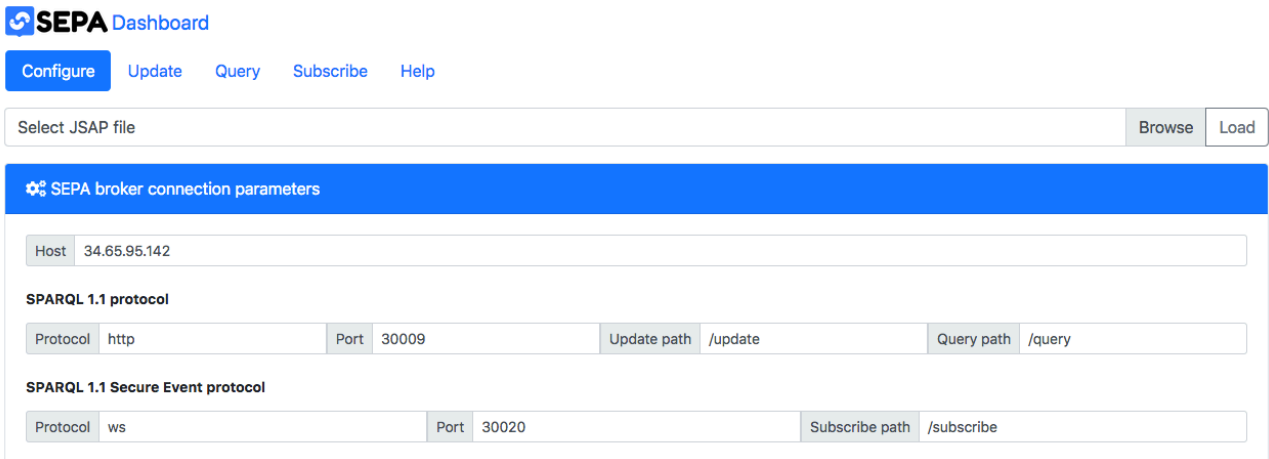


Figura 30 Pagina di configurazione del SEPA

Ora si inseriscono per esempio degli utenti andando nella sezione “update” > “register\_users”.

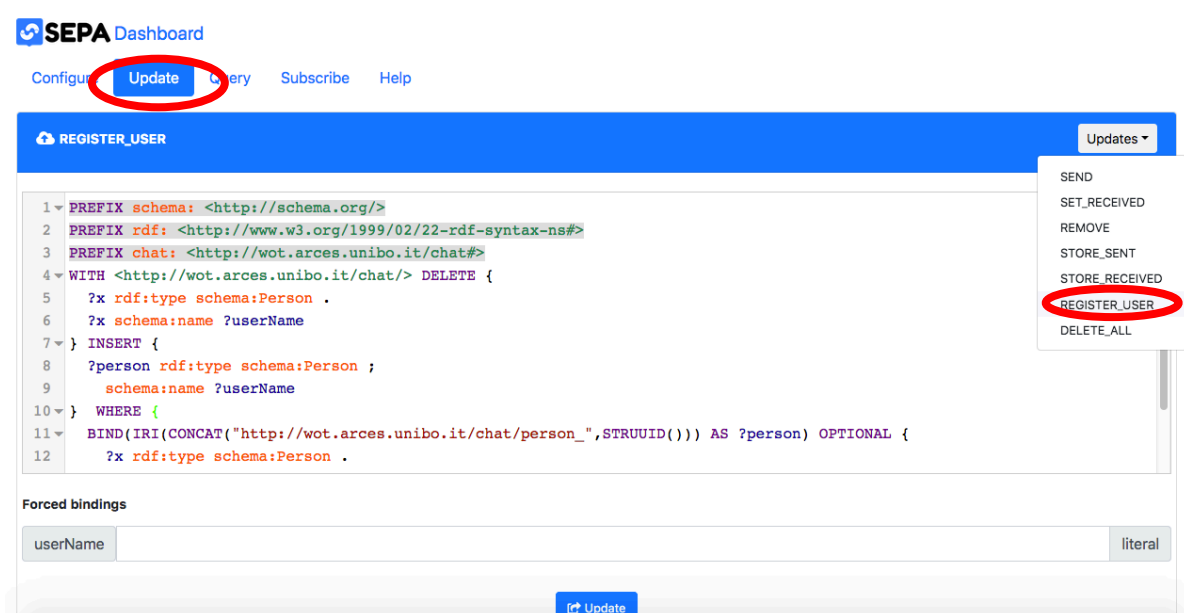


Figura 31 Modalità di inserimento di nuovi dati

Poi si verifica che nelle sezioni “subscribe” e “query” ci siano i risultati degli inserimenti precedentemente effettuati. E’ importante che la verifica venga fatta in entrambe le voci così che vengano testate le due porte, e quindi le relative esposizioni sulla rete utilizzate dal SEPA.



```
1 PREFIX schema: <http://schema.org/>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 PREFIX chat: <http://wot.arces.unibo.it/chat#>
4 SELECT ?user ?userName WHERE {
5   GRAPH <http://wot.arces.unibo.it/chat/> {
6     ?user rdf:type schema:Person ;
7         schema:name ?userName
8   }
9 }
```

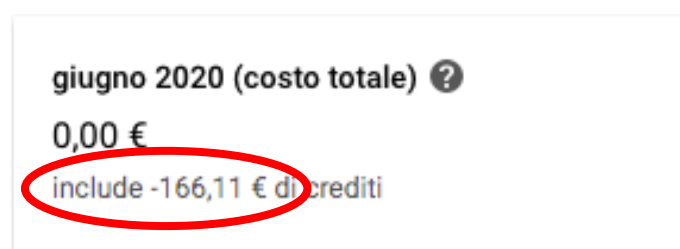
Figura 32 Verifica degli utenti inseriti precedentemente

Infine rigenerando il pod db (eliminando il pod dalla dashboard, esso si rigenera automaticamente) e, tornando sul SEPA, si vedrà che non sono stati persi i dati inseriti precedentemente, cioè quelli salvati dal vecchio pod sul volume permanente.

## Costi

L'utilizzo di Google Kubernetes Engine (GKE) non è gratuito a meno di una prova iniziale di 12 mesi e del valore di 300€. Di seguito verrà riportato il costo di questa esperienza. Il costo si basa prettamente sulle attività svolte: memoria utilizzata, quantità di cpu richiesta, localizzazione del cluster, durata dell'esistenza del cluster sulla piattaforma.

Un mese di attività di questo specifico sistema di prova sulle piattaforme Google è costato 166,11€.



giugno 2020 (costo totale) ?  
0,00 €  
include -166,11 € di crediti

Figura 33 Spesa di crediti nel mese di Giugno

Chiaramente questi crediti vengono sottratti dai 300 iniziali di prova gratuita, infatti il nostro account di fatturazione non ha avuto spese come attesta anche la mail inviata da Google all'indirizzo di fatturazione.

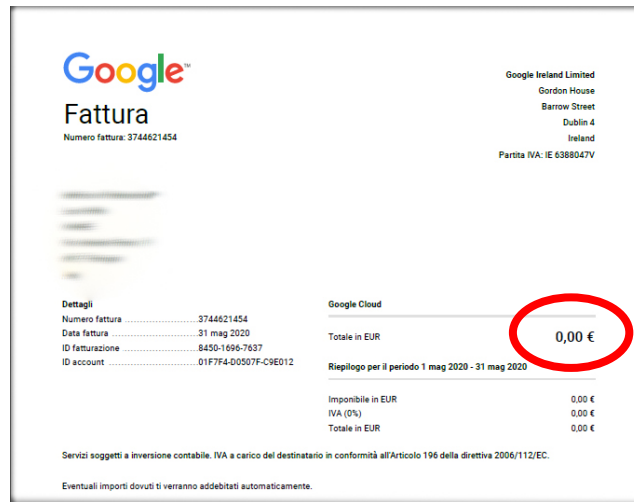


Figura 34 Mail contenente la fattura del nostro account

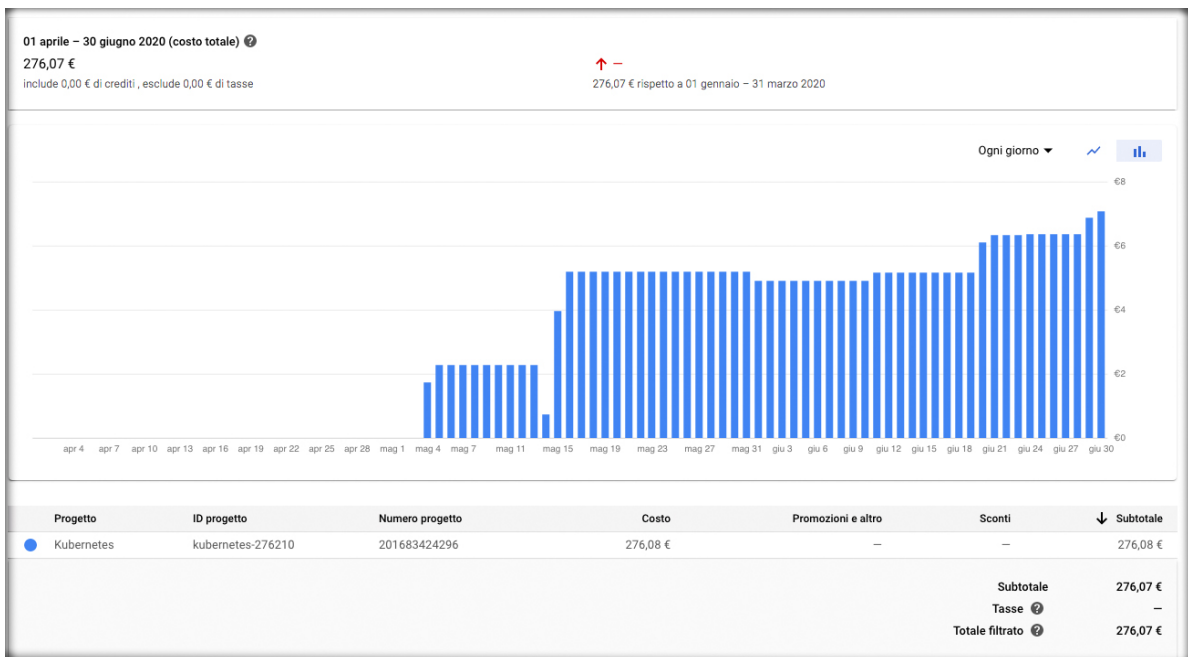


Figura 35 Andamento giornaliero delle spese

# SWAMP su Google Kubernetes Engine



## Introduzione a SWAMP

SWAMP (Smart Water Management Platform) [24] è una piattaforma che si occupa di gestire al meglio le risorse idriche in agricoltura. Questa piattaforma si basa su: IoT (Internet Of Things), analisi dei dati, dispositivi autonomi e altre tecnologie correlate. Lo scopo principale è quello di creare un sistema di irrigazione intelligente ad alta precisione, che distribuisce la giusta quantità di acqua alle colture senza eccedere né scarseggiare.

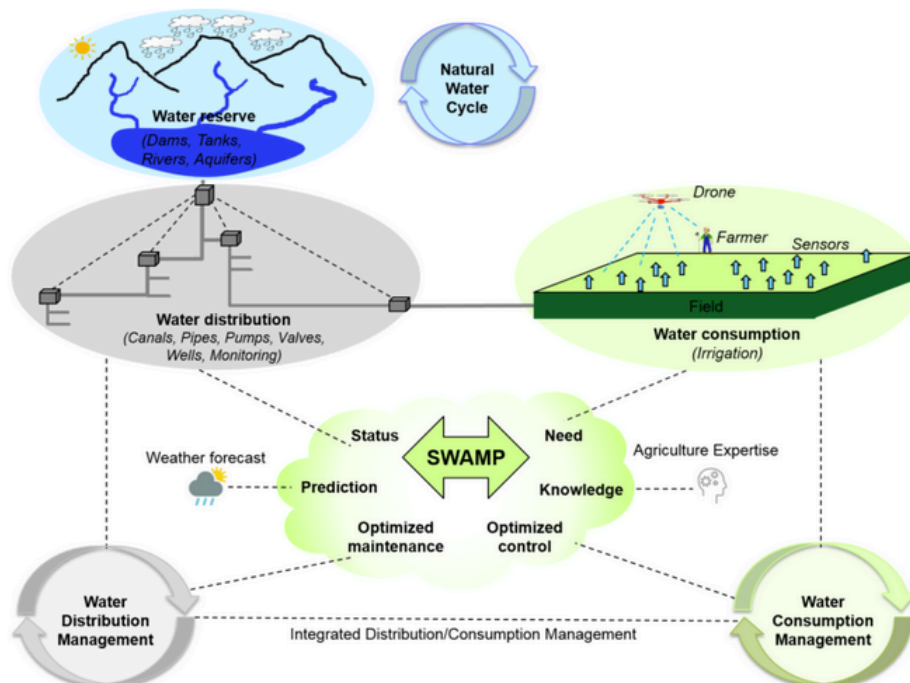
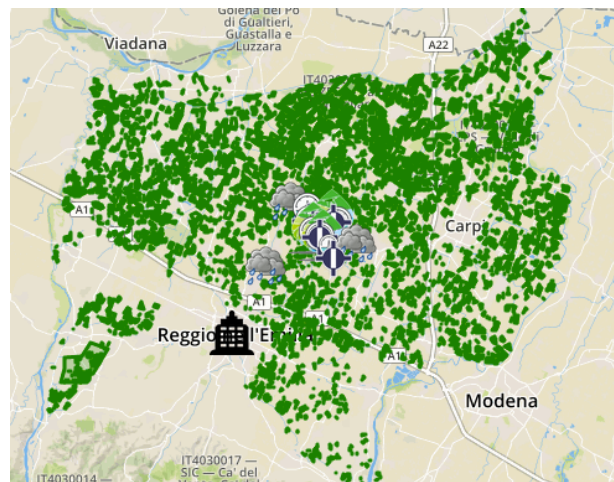


Figura 36 Descrizione del progetto SWAMP (fonte [26])

Per fare questo si utilizza una serie di sensori e droni, come per esempio: sensori per prelevare la quantità di acqua nei canali e sensori

per l'umidità presente nel terreno. Si intrecciano i dati con altre informazioni, come le previsioni meteo e dati ARPAE Emilia-Romagna [26] sull'umidità dei terreni, e infine si ottiene la quantità giusta di acqua da distribuire.

Questa tecnologia al momento è stata implementata nelle zone limitrofe a Reggio Emilia dove sono stati mappati tutti i campi con le relative colture.



*Figura 37 Mappa che mostra le richieste di irrigazioni per i vari campi nella zona del pilot Italiano di Reggio Emilia*

Per accedere in modo diretto a tutti i dati monitorati si deve consultare il sito SWAMP al link [25]. In questo sito si possono vedere: tutti i campi presenti (Figura 37), e il loro utilizzo, le previsioni meteo in tempo reale, la quantità di acqua negli acquedotti e tutte le informazioni necessarie al sistema. Questo sito si basa sulla tecnologia del SEPA, in questo modo si possono eseguire delle query per conoscere i dati raccolti e anche lo storico.



Feature of interest

S. Soc. Agricola

Observation **Display time** UTC **Time zone:** UTC

**Precipitation** 0 mm July 10, 2020 9:00 PM [History](#) [Compare](#)

**Air Temperature** 22.21 degC July 10, 2020 10:00 PM [History](#) [Compare](#)

Forecast

Today (Saturday, July 11th 2020)

LAI (#) 3.49 [History](#)

**Irrigation needs (mm)** 0.00 [History](#)

Tomorrow (Sunday, July 12th 2020)

LAI (#) 3.49 [History](#)

**Irrigation needs (mm)** 0.00 [History](#)

Figura 38 Pagina Web in cui si vedono le condizioni meteo e la quantità di acqua necessaria per irrigare questo campo

## Sistema SWAMP

La Figura 39 mostra tutti i container presenti nello sviluppo di SWAMP su GKE. Essa, inoltre, fa vedere anche tutti i collegamenti tra i vari pod.

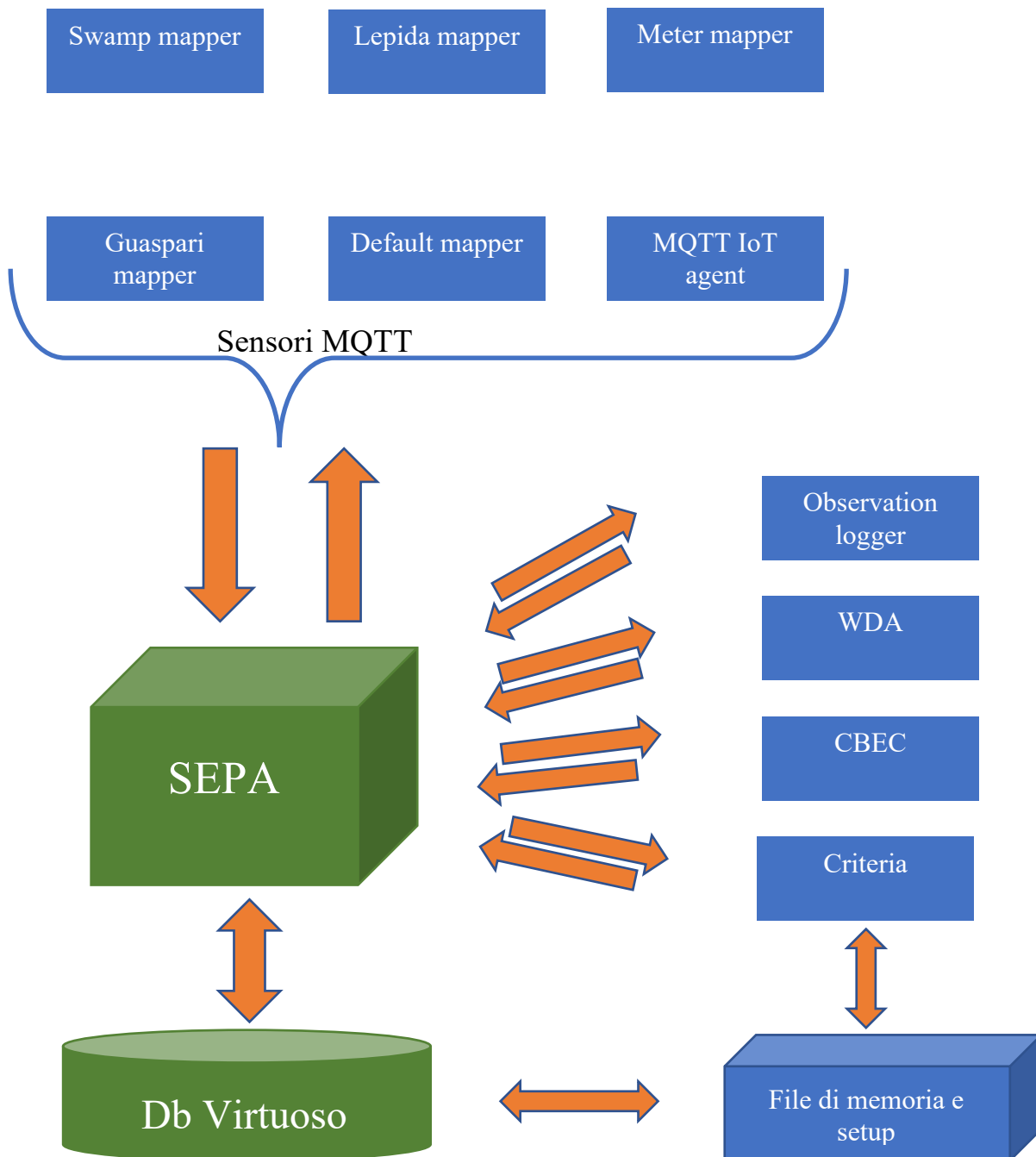


Figura 39 Schema riassuntivo delle componenti del sistema SWAMP

Di seguito vengono riportati tutti i software presenti nel sistema con i relativi riferimenti Docker Hub e Git Hub.

Componente	Link Docker Hub
Applicazione MQTT che si occupa di prelevare i dati dei sensori provenienti da varie fonti. L'applicazione si collega a vari broker MQTT e consente di mappare i messaggi in dati semantici. GitHub: <a href="https://github.com/arces-wot/SEPA-MqttIoTAgent">https://github.com/arces-wot/SEPA-MqttIoTAgent</a>	
MQTT IoT Agent	<a href="https://hub.docker.com/r/lroffia/swamp-mqttiotagent">https://hub.docker.com/r/lroffia/swamp-mqttiotagent</a>
SWAMP mapper	<a href="https://hub.docker.com/r/lroffia/swamp-swampmapper">https://hub.docker.com/r/lroffia/swamp-swampmapper</a>
METER mapper	<a href="https://hub.docker.com/r/lroffia/swamp-metermapper">https://hub.docker.com/r/lroffia/swamp-metermapper</a>
LEPIDA mapper	<a href="https://hub.docker.com/r/lroffia/swamp-lepidamapper">https://hub.docker.com/r/lroffia/swamp-lepidamapper</a>
Default mapper	<a href="https://hub.docker.com/r/lroffia/swamp-defaultmapper">https://hub.docker.com/r/lroffia/swamp-defaultmapper</a>
Guaspaeri mapper	<a href="https://hub.docker.com/r/lroffia/swamp-guasparimapper">https://hub.docker.com/r/lroffia/swamp-guasparimapper</a>
Storicizza i dati GitHub: <a href="https://github.com/arces-wot/SEPA-MqttIoTAgent">https://github.com/arces-wot/SEPA-MqttIoTAgent</a>	
Observation Logger	<a href="https://hub.docker.com/r/lroffia/swamp-observationlogger">https://hub.docker.com/r/lroffia/swamp-observationlogger</a>
Il motore del sistema GitHub: <a href="https://github.com/arces-wot/SEPA">https://github.com/arces-wot/SEPA</a>	
SEPA	<a href="https://hub.docker.com/r/alessandramanzelli/engine">https://hub.docker.com/r/alessandramanzelli/engine</a>
Il database a grafo GitHub: <a href="https://github.com/openlink/virtuoso-opensource">https://github.com/openlink/virtuoso-opensource</a>	
DB VIRTUOSO	<a href="https://hub.docker.com/r/tenforce/virtuoso">https://hub.docker.com/r/tenforce/virtuoso</a>
Si occupa di prelevare i dati di irrigazione dal DB gestito dal CBEC (Consorzio Bonifica Emilia Centrale) GitHub: <a href="https://github.com/arces-wot/swamp-cbec">https://github.com/arces-wot/swamp-cbec</a>	
CBEC adapter	<a href="https://hub.docker.com/r/lroffia/swamp-cbec-irrigation">https://hub.docker.com/r/lroffia/swamp-cbec-irrigation</a>
E' l'applicazione Web per l'utente finale GitHub: <a href="https://github.com/arces-wot/SEPAview">https://github.com/arces-wot/SEPAview</a>	
WDA (Water Distribution Application)	<a href="https://hub.docker.com/r/lroffia/swamp-wda">https://hub.docker.com/r/lroffia/swamp-wda</a>
Implementa il modello di bilancio idrico del suolo ed è fornita da ARPAE [28] GitHub: <a href="https://github.com/arces-wot/CriteriaSWAMPService">https://github.com/arces-wot/CriteriaSWAMPService</a>	
CRITERIA	<a href="https://hub.docker.com/r/lroffia/swamp-criteria">https://hub.docker.com/r/lroffia/swamp-criteria</a>

Tabella 1 Docker containers dell'applicazione SWAMP

Di seguito vengono riportati i file yaml utilizzati per creare i deployment dei relativi software. Non vengono riportati quello del db e quello di SEPA, in quanto sono uguali a quelli riportati nei capitoli precedenti.

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: swampmapper
5    labels:
6      app: swampmapper
7  spec:
8    selector:
9      matchLabels:
10     app: swampmapper
11   replicas: 1
12   template:
13     metadata:
14       labels:
15         app: swampmapper
16     spec:
17       containers:
18         - name: swamp-swampmapper
19           image: lroffia/swamp-swampmapper:0.3
20           ports:
21             - containerPort: 8000
22             protocol: TCP

```

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: observationlogger
5    labels:
6      app: observationlogger
7  spec:
8    selector:
9      matchLabels:
10     app: observationlogger
11   replicas: 1
12   template:
13     metadata:
14       labels:
15         app: observationlogger
16     spec:
17       containers:
18         - name: swamp-observationlogger
19           image: lroffia/swamp-observationlogger:0.3
20           ports:
21             - containerPort: 8000
22             protocol: TCP

```

Figura 40 File YAML per il deployment di SWAMP mapper (sinistra), file YAML per il deployment di Observation logger (destra)

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: mqttiotagent
5    labels:
6      app: mqttiotagent
7  spec:
8    selector:
9      matchLabels:
10     app: mqttiotagent
11   replicas: 1
12   template:
13     metadata:
14       labels:
15         app: mqttiotagent
16     spec:
17       containers:
18         - name: swamp-mqttiotagent
19           image: lroffia/swamp-mqttiotagent:0.3
20           ports:
21             - containerPort: 8000
22             protocol: TCP

```

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: metermapper
5    labels:
6      app: metermapper
7  spec:
8    selector:
9      matchLabels:
10     app: metermapper
11   replicas: 1
12   template:
13     metadata:
14       labels:
15         app: metermapper
16     spec:
17       containers:
18         - name: swamp-metermapper
19           image: lroffia/swamp-metermapper:0.3
20           ports:
21             - containerPort: 8000
22             protocol: TCP

```

Figura 41 File YAML per il deployment di MQTT Agent (sinistra), file YAML per il deployment di METER mapper (destra)

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: lepidamapper
5    labels:
6      app: lepidamapper
7  spec:
8    selector:
9      matchLabels:
10     app: lepidamapper
11   replicas: 1
12   template:
13     metadata:
14       labels:
15         app: lepidamapper
16     spec:
17       containers:
18         - name: swamp-lepidamapper
19           image: lroffia/swamp-lepidamapper:0.3
20           ports:
21             - containerPort: 8000
22             protocol: TCP

```

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: guasparimapper
5    labels:
6      app: guasparimapper
7  spec:
8    selector:
9      matchLabels:
10     app: guasparimapper
11   replicas: 1
12   template:
13     metadata:
14       labels:
15         app: guasparimapper
16     spec:
17       containers:
18         - name: swamp-guasparimapper
19           image: lroffia/swamp-guasparimapper:0.3
20           ports:
21             - containerPort: 8000
22             protocol: TCP

```

Figura 42 File YAML per il deployment di LEPIDA mapper (sinistra), file YAML per il deployment di Guaspari mapper (destra)

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: defaultmapper
5    labels:
6      app: defaultmapper
7  spec:
8    selector:
9      matchLabels:
10     app: defaultmapper
11   replicas: 1
12   template:
13     metadata:
14       labels:
15         app: defaultmapper
16     spec:
17       containers:
18         - name: swamp-defaultmapper
19           image: lroffia/swamp-defaultmapper:0.3
20           ports:
21             - containerPort: 8000
22             protocol: TCP

```

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: criteria
5    labels:
6      app: criteria
7  spec:
8    selector:
9      matchLabels:
10     app: criteria
11   replicas: 1
12   template:
13     metadata:
14       labels:
15         app: criteria
16     spec:
17       containers:
18         - name: swamp-criteria
19           image: lroffia/swamp-criteria:0.3
20           ports:
21             - containerPort: 8000
22             protocol: TCP

```

Figura 43 File YAML per il deployment di Default mapper (sinistra), file YAML per il deployment di CRITERIA (destra)

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: cbec-irrigation
5    labels:
6      app: cbec-irrigation
7  spec:
8    selector:
9      matchLabels:
10       app: cbec-irrigation
11   replicas: 1
12   template:
13     metadata:
14       labels:
15         app: cbec-irrigation
16     spec:
17       containers:
18         - name: swamp-cbec-irrigation
19           image: lroffia/swamp-cbec-irrigation:0.3
20           ports:
21             - containerPort: 8000
22               protocol: TCP

```

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: wda
5    labels:
6      app: wda
7  spec:
8    selector:
9      matchLabels:
10       app: wda
11   replicas: 1
12   template:
13     metadata:
14       labels:
15         app: criteria
16     spec:
17       containers:
18         - name: swamp-wda
19           image: lroffia/swamp-wda:0.3
20           ports:
21             - containerPort: 8000
22               protocol: TCP

```

Figura 44 File YAML per il deployment di CBEC adapter (sinistra), file YAML per il deployment di WDA (destra)

Di seguito vengono riportati i file yaml utilizzati per creare i service dei relativi pod del sistema SWAMP anche in questo caso non vengono riportati quello di SEPA e Virtuoso in quanto sono uguali a quelli trattati nel capitolo precedente.

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: swampmapper
5    labels:
6      app: swampmapper
7  spec:
8    type: NodePort
9    ports:
10   - name: mqtt-8000
11     targetPort: 8000
12     port: 8000
13     nodePort: 30003
14     protocol: TCP
15   - name: mqtt-9000
16     targetPort: 9000
17     port: 9000
18     nodePort: 30004
19     protocol: TCP
20   selector:
21     app: swampmapper

```

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: observationlogger
5    labels:
6      app: observationlogger
7  spec:
8    type: NodePort
9    ports:
10   - name: mqtt-8000
11     targetPort: 8000
12     port: 8000
13     nodePort: 30005
14     protocol: TCP
15   - name: mqtt-9000
16     targetPort: 9000
17     port: 9000
18     nodePort: 30006
19     protocol: TCP
20   selector:
21     app: observationlogger

```

Figura 45 File YAML per il service di SWAMP mapper (sinistra), file YAML per il service di Observation logger (destra)

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: mqttiotagent
5    labels:
6      app: mqttiotagent
7  spec:
8    type: NodePort
9    ports:
10   - name: mqtt-8000
11     targetPort: 8000
12     port: 8000
13     nodePort: 30001
14     protocol: TCP
15   - name: mqtt-9000
16     targetPort: 9000
17     port: 9000
18     nodePort: 30002
19     protocol: TCP
20   selector:
21     app: mqttiotagent

```

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: metermapper
5    labels:
6      app: metermapper
7  spec:
8    type: NodePort
9    ports:
10   - name: mqtt-8000
11     targetPort: 8000
12     port: 8000
13     nodePort: 30031
14     protocol: TCP
15   - name: mqtt-9000
16     targetPort: 9000
17     port: 9000
18     nodePort: 30032
19     protocol: TCP
20   selector:
21     app: metermapper

```

Figura 46 File YAML per il service di MQTT IoT Agent (sinistra), file YAML per il service di METER mapper (destra)

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: lepidamapper
5    labels:
6      app: lepidamapper
7  spec:
8    type: NodePort
9    ports:
10   - name: lepidamapper-8000
11     targetPort: 8000
12     port: 8000
13     nodePort: 30033
14     protocol: TCP
15   - name: lepidamapper-9000
16     targetPort: 9000
17     port: 9000
18     nodePort: 30034
19     protocol: TCP
20   selector:
21     app: lepidamapper

```

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: guasparimapper
5    labels:
6      app: guasparimapper
7  spec:
8    type: NodePort
9    ports:
10   - name: guasparimapper-8000
11     targetPort: 8000
12     port: 8000
13     nodePort: 30035
14     protocol: TCP
15   - name: guasparimapper-9000
16     targetPort: 9000
17     port: 9000
18     nodePort: 30036
19     protocol: TCP
20   selector:
21     app: guasparimapper

```

Figura 47 File YAML per il service di LEPIDA mapper (sinistra), file YAML per il service di Guaspari mapper (destra)

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: defaultmapper
5    labels:
6      app: defaultmapper
7  spec:
8    type: NodePort
9    ports:
10   - name: defaultmapper-8000
11     targetPort: 8000
12     port: 8000
13     nodePort: 30037
14     protocol: TCP
15   - name: defaultmapper-9000
16     targetPort: 9000
17     port: 9000
18     nodePort: 30038
19     protocol: TCP
20   selector:
21     app: defaultmapper

```

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: criteria
5    labels:
6      app: criteria
7  spec:
8    type: NodePort
9    ports:
10   - name: criteria-8000
11     targetPort: 8000
12     port: 8000
13     nodePort: 30041
14     protocol: TCP
15   - name: criteria-9000
16     targetPort: 9000
17     port: 9000
18     nodePort: 30042
19     protocol: TCP
20   selector:
21     app: criteria

```

Figura 48 File YAML per il service di Default mapper (sinistra), file YAML per il service di CRITERIA (destra)

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: cbec-irrigation
5    labels:
6      app: cbec-irrigation
7  spec:
8    type: NodePort
9    ports:
10   - name: cbec-irrigation-8000
11     targetPort: 8000
12     port: 8000
13     nodePort: 30039
14     protocol: TCP
15   - name: cbec-irrigation-9000
16     targetPort: 9000
17     port: 9000
18     nodePort: 30040
19     protocol: TCP
20   selector:
21     app: cbec-irrigation

```

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: wda
5    labels:
6      app: wda
7  spec:
8    type: NodePort
9    ports:
10   - name: wda-8000
11     targetPort: 8000
12     port: 8000
13     nodePort: 30043
14     protocol: TCP
15   - name: wda-9000
16     targetPort: 9000
17     port: 9000
18     nodePort: 30044
19     protocol: TCP
20   selector:
21     app: wda

```

Figura 49 File YAML per il service di CBEC adapter (sinistra), file YAML per il service di WDA (destra)



## Modalità di accesso remoto a file condivisi

Si possono trovare diverse soluzioni per poter accedere a dei registri di memoria permanenti per salvare e mantenere alcuni dati utili alle applicazioni contenute nei pod.

Il primo di quelli provati in questa tesi è utilizzare i volumi permanenti e le relative richieste di volumi permanenti (persistent volume claim). In questo modo il modo il pod si collega al volume (che può avere capienza variabile a seconda del necessario), e lì salva i suoi dati, che rimarranno anche qualora il pod dovesse essere rigenerato. Questo caso è stato trattato nel paragrafo Volumi di memoria.

Seconda possibilità è quella di utilizzare dei persistent disk. Ciò è simile all'utilizzo delle istanze nel filestore, ma permette di risparmiare molto a livello economico in quanto, una istanza minima è di un terabyte, mentre i disk possono essere della capienza che si sceglie nel momento della sua creazione. In questo caso bisogna creare un NFS (Network File Sistem) per permettere ai pod di accedere ai dischi. Come primo passaggio bisogna creare un disco, o tramite dashboard oppure tramite il comando: `gcloud compute disks create --size=<dimensione del disco> --zone=<zona geografica del cluster> gce-nfs-disk`. In seguito si deve creare il server NFS, il relativo service, persistent volume e persistent volume claim con file yaml visibili al link [28].

Infine un'altra opzione è quella di utilizzare gli FTP container. Si deve creare una un volume persistente e una volume persistent claim e la si associa a questo container il quale fa solo da tramite grazie alla rete tra il locale e questo volume.

## Ringraziamenti

Proprio in un periodo così particolare si concludono questi anni, un'esperienza unica che mi ha permesso di maturare e crescere come persona oltre che come studente.

Questi quattro anni si concludono con un lavoro che non pensavo mi potesse interessare così tanto. Sono stato contento di aver partecipato a questo progetto e per questo devo ringraziare il professore Luca Roffia per la sua disponibilità e comprensione, il correlatore ing. Cristiano Aguzzi e Alessandra Manzelli, che mi ha preceduto in questo lavoro, e mi ha sempre saputo dare consigli importanti.

Poi vorrei ringraziare la mia famiglia (anche chi non c'è più e avrebbe tanto voluto vivere questo momento), perché mi ha sempre supportato e sopportato nei momenti di difficoltà ed è sempre stata orgogliosa di me. Inoltre devo ringraziare tutti i miei amici: compagni di corso e non, amici di grandi studiate e di grandi suonate, con i quali ho passato tanti momenti felici che hanno reso questi ultimi anni un po' più leggeri, felici e talvolta spensierati. Infine, ma non per importanza, un grazie anche alla mia "seconda famiglia": Renzo, Sabina e tutta la truppa del bagno Paola, che mi hanno sempre permesso di andare avanti con gli esami nonostante il lavoro.

## Bibliografia e sitografia

- [1] Alessandra Manzelli, “Orchestrazione di containers per una piattaforma di distribuzione di dati semantici”, Tesi di Laurea, ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA- CAMPUS DI CESENA, SCUOLA DI INGEGNERIA E ARCHITETTURA, CORSO DI LAUREA IN INGEGNERIA ELETTRONICA PER L’ENERGIA E L’INFORMAZIONE, A.A. 2018-19.
- [2] Cos’è Kubernetes?, documentazione:  
<https://kubernetes.io/it/docs/concepts/overview/what-is-kubernetes/>
- [3] Kubernetes, Orchestrazione di container in produzione:  
<https://kubernetes.io/it/>
- [4] Documentazione Kubernetes, Standardized Glossary  
<https://kubernetes.io/it/docs/reference/glossary/?fundamental=true>
- [5] Docker: <https://www.docker.com/>
- [6] Docker Hub: <https://hub.docker.com/>
- [7] VIRTUOSO: <https://virtuoso.openlinksw.com/>
- [8] SEPA: <https://github.com/arces-wot/SEPA>
- [9] Kubernetes, documentation, concept: <https://kubernetes.io/docs/concepts/>
- [10] Kubernetes, documentatio deployment :  
<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>
- [11] Kubernetes, documentation pod overview  
<https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/>
- [12] Kubernetes, documentation service:  
<https://kubernetes.io/docs/concepts/services-networking/service/>
- [13] Kubernetes, documentation using minikube to create a cluster:  
<https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/>
- [14] Kubernetes, documentation viewing pods end nodes:  
<https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/>

- [15] Kubernetes, documentation using a service to expose your app:  
<https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/>
- [16] Kubernetes, documentation installing kubernetes with minikube:  
<https://kubernetes.io/docs/setup/learning-environment/minikube/>
- [17] Kubernetes, documentation install Minikube:  
<https://kubernetes.io/docs/tasks/tools/install-minikube/>
- [18] I servizi di Google Cloud Platform, spiegati bene. Quali sono gli ambiti di utilizzo più comuni per il cloud e quali le componenti di Google Cloud Platform (GCP) necessarie per sfruttarli al meglio? Ve lo spieghiamo noi:  
[https://www.cwi.it/cio/servizi-google-cloud-platform\\_42110382](https://www.cwi.it/cio/servizi-google-cloud-platform_42110382)
- [19] Google Kubernetes Engine: <https://cloud.google.com/kubernetes-engine#all-features>
- [20] Visione della console sul progetto kubernetes-276210 utilizzato per la prova:  
<https://console.cloud.google.com/kubernetes/add?project=kubernetes-276210>
- [21] Google Cloud, exposing applications using services:  
<https://cloud.google.com/kubernetes-engine/docs/how-to/exposing-apps>
- [22] Google Cloud, persistent volume with persistent disk:  
<https://cloud.google.com/kubernetes-engine/docs/concepts/persistent-volumes>
- [23] SEPA Dashboard: <http://mml.arces.unibo.it/apps/dashboard/>
- [24] Progetto SWAMP: <http://swamp-project.org/about/>
- [25] Visualizzazione dei dati del progetto SWAMP:  
<http://mml.arces.unibo.it/swamp/wda/>
- [26] Sito ARPAE Emilia-Romagna: <https://www.arpae.it/>
- [27] ARPAE Emilia-Romagna, CRITERIA modello di bilancio idrico:  
[https://www.arpae.it/dettaglio\\_documento.asp?id=708&idlivello=64](https://www.arpae.it/dettaglio_documento.asp?id=708&idlivello=64)
- [28] NFS persistent volumes with Kubernetes on GKE – A case study:  
<https://medium.com/platformer-blog/nfs-persistent-volumes-with-kubernetes-a-case-study-ce1ed6e2c266>