Foggy Day

GIANMARCO BENCIVENNI, SALVATORE FODERARO, and ALESSANDRO PONTIS, Università di Roma Torvergata

Con questo articolo si presenta il progetto elaborato per il Corso di Sistemi Distribuiti e Cloud Computing presso l'Università di Roma Torvergata, nell'anno accademico 2019/2020.

L'applicazione si fonda sul paradigma del Fog Computing, ove la caratteristica saliente risiede nella circoscrizione del traffico di rete presso le vicinanze dell'utente, così come dei dispositivi IoT utilizzati.

Foggy Day prende posizione in ambito agricolo, come applicazione di Precision Farming a supporto delle coltivazioni miste e di variabile estensione. Lo scopo dell'applicazione è quello di fornire supporto alla gestione delle risorse agricole, ottimizzando i costi e mantenendo la qualità dei prodotti: per fare ciò, si è fatto uso di sensori IoT per il rilevamento dei parametri e di immagini del suolo coltivato, nonché di dati esterni, provenienti da stazioni meteorologiche e da immagini satellitari. Una fase di Analisi e Pianificazione viene effettuata giornalmente, a partire dai dati raccolti e collezionati, per produrre un piano di esecuzione da comunicare ai dispositivi IoT esecutori.

Additional Key Words and Phrases: IoT, Internet of Things, Fog Computing, Edge Computing, Precision Farming, FoggyDay, Cloud Computing, sensor, agriculture, MAPE, monitoring, REST API, microservices

Authors' address: Gianmarco Bencivenni, gimbors95@gmail.com; Salvatore Foderaro, salvatore.foderaro@gmail.com; Alessandro Pontis, alessandropontis95@gmail.com, Università di Roma Torvergata, Via del Politecnico 1, Roma, Italy.

XXXX-XXXX/2020/9-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Gianmarco Bencivenni, Salvatore Foderaro, and Alessandro Pontis. 2020. Foggy Day. 1, 1 (September 2020), 17 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 SPECIFICA ED ANALISI DEI REQUISITI

In questa sezione vengono riportate le specifiche dei requisiti (funzionali e non funzionali) che il prodotto finale è tenuto a soddisfare.

1.1 Requisiti funzionali

Per quanto concerne la specifica dei requisiti funzionali, ci limitiamo a definire le principali funzionalità a cui mira l'applicativo, trattandosi a tutti gli effetti di un *"progetto libero"*, a scopo didattico.

In riferimento alla tabella sottostante, l'applicazione:

Table 1. Tabella dei requisiti funzionali.

Importanza	Requisito
Deve	Fornire un supporto automatizzato alla gestione delle risorse idriche per l'irrigazione
Deve	In base a previsioni e ad accurata pianificazione, minimizzare lo spreco d'acqua
Deve	In base a previsioni e ad accurata pianificazione, minimizzare i costi di gestione
Deve	Esporre un'interfaccia utente per la visualizzazione e il controllo del sistema
Dovrebbe	Interagire automaticamente con l'utente, per segnalare guasti nel sistema

1.2 Requisiti non funzionali

Si procede ora a riportare i requisiti non funzionali, come da specifica di assegnazione del progetto.

L'applicazione:

(1) Deve essere distribuita su molteplici nodi.
(2) Deve utilizzare almeno un servizio Cloud per eseguire una funzionalità dell'applicazione che sia computazionalmente onerosa.
(3) Deve essere scalabile e tollerante a crash dei nodi Fog o Cloud su cui l'applicazione viene eseguita.
(4) Dovrebbe utilizzare un protocollo di messaggistica per IoT.

Si parla a tutti gli effetti di requisiti non funzionali: i requisiti 1, 2 e 4 sono *requisiti di processo* riguardanti l'implementazione, mentre il requisito 3 è un *requisito di prodotto* riguardante affidabilità e prestazioni.

Di seguito, si riportano le scelte intraprese in fase di progettazione e di sviluppo per coniugare la realizzazione dell'applicativo introdotto con il soddisfacimento dei requisiti indicati.

(1) L'utilizzo di dispositivi IoT, quali i sensori utilizzati per la raccolta dei dati di temperatura e umidità del suolo, o quelli per regolare l'apertura e la chiusura meccanica delle valvole idriche di irrigazione, suggerisce già da sé la natura distribuita dell'applicazione software su molteplici nodi, fisicamente distinti.
In aggiunta a ciò, il core dell'applicazione è costituito da molteplici moduli software eseguibili all'interno di containers isolati. L'utilizzo della tecnologia del Container, supportata da un adeguato tool d'orchestrazione, permette la distribuzione del software su uno o più nodi, assicurando di pari passo isolamento e portabilità.
Il tema della distribuzione del software verrà approfondito in dettaglio nella sezione 2, dedicata al design e all'architettura dell'applicazione.

(2) Un modulo software sarà responsabile dell'analisi dei dati raccolti e della produzione di un conseguente piano giornaliero di gestione delle risorse agricole. Sarà questo modulo ad essere eseguito su una piattaforma Cloud, in modo da alleggerire i nodi Fog dalla computazione.
Un altro servizio Cloud utilizzato è quello di Storage: pur essendo previsto il mantenimento di un database locale ai nodi Fog, è stato scelto di salvare uno storico per dati raccolti nei giorni passati.
(3) Un'analisi sulla tolleranza e sulla scalabilità dell'applicazione verrà effettuata nelle sezioni successive.
(4) La struttura dell'applicazione ricalca il paradigma di progettazione per responsabilità (RDD), dove ogni modulo (o *famiglia di moduli*) espone un servizio in un dominio ben definito e limitato, a favore degli altri moduli che lo richiedono.
La comunicazione tra questi è basata su REST API, ove i parametri necessari all'esecuzione dei servizi vengono inclusi in richieste HTTP. In questo contesto, si parla dunque di *comunicazione sincrona* e di *accoppiamento forte*, essendo che le due parti dovranno sempre esser compresenti nello scambio di informazioni, e trattandosi perlopiù di chiamate bloccanti.

2 DESIGN E ARCHITETTURA DELL'APPLICAZIONE

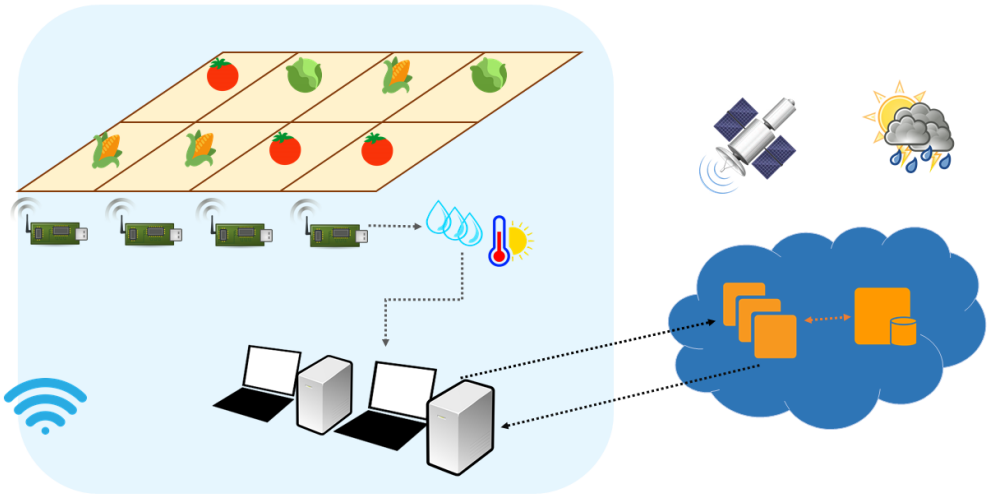

Fig. 1. Schema ad alto livello dell'architettura dell'applicazione.

2.1 Architettura software

La comunicazione tra i singoli componenti software dell'applicazione segue lo stile architetturale *Restful style*: lo scambio di informazioni, oltre ad avvenire esclusivamente tramite richieste HTTP, è *stateless* e privo di sessioni. L'architettura del software si basa sul paradigma dell'*Autonomic Computing*, secondo il pattern MAPE, dove i componenti si alternano nelle fasi seguenti:

- Monitor
- Analyze

- Plan
- Execute

In particolare, il sistema effettua cicli periodici di auto-configurazione (*self-configuring*), a partire dai dati raccolti nella fase di monitoraggio, passando per le fasi di analisi e pianificazione, fino a giungere alla fase finale d'esecuzione, in cui vengono messi in atto i comandi segnalati dal planner, in reazione alle sintomatologie valutate.

Il pattern architetturale MAPE viene qui decentralizzato, assumendo una configurazione gerarchica di tipo *Master-Worker*: i moduli di monitoraggio e di esecuzione vengono eseguiti su nodi distinti, mentre viene mantenuto unico il nodo che opera le mansioni di analisi dei dati e di pianificazione. In questo modo, si viene a designare un Master-node che detiene una visione globale rispetto alle modifiche che dovrà apportare al sistema.

Per quanto il singolo nodo Master possa risultare un collo di bottiglia, la decisione di eseguire il nodo su piattaforma Cloud fornisce garanzie: all'aumentare delle aree di coltivazione che fanno riferimento allo stesso nodo Master, sarà compito della piattaforma Cloud di gestire il numero di repliche scalando orizzontalmente, e garantire tolleranza ai guasti eventuali.

Di seguito, si descrivono nel dettaglio le funzionalità dell'applicazione, quindi i ruoli dei singoli componenti che cooperano nel ciclo MAPE.

- Monitoraggio

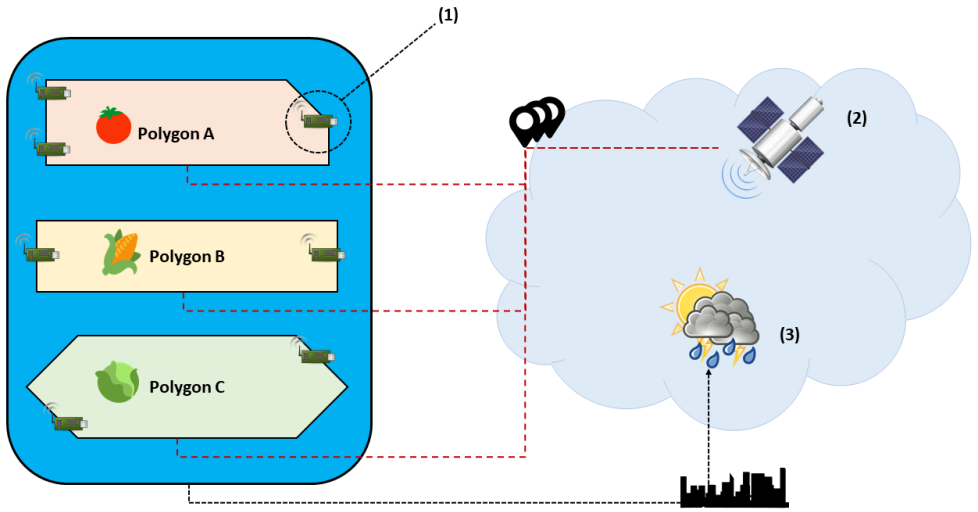

Fig. 2. Schema delle tre tipologie di Monitor-nodes in relazione alle coltivazioni.

La mansione di monitoraggio viene effettuata su più fronti.

(1) In primo luogo, i sensori IoT che prelevano i parametri fisici di temperatura e umidità del suolo svolgono la funzione di Monitor. Ogni singolo sensore appartiene ad un poligono, che è delimitato dalle coordinate di latitudine e longitudine dei propri vertici, e identifica una specifica coltivazione.
I sensori rilevano dunque una parte dei parametri necessari alla computazione del planning, e inviano, ad intervalli regolari, nuovi valori al modulo responsabile del collezionamento,

situato nel cluster Fog.

(2) La seconda tipologia di Monitor-node risiede nel Cloud: si tratta di un servizio esterno all'applicativo, accessibile grazie all'esposizione di API *ad hoc*, che fornisce dati relativi a parametri fisici provenienti da immagini ottiche delle missioni satellitari americana *Landsat 8 [1]* ed europea *ESA Copernicus - Sentinel 2 [2]*.
I dati raccolti fanno riferimento ad immagini reali dei poligoni specificati, e i parametri di maggiore interesse sono i valori degli indici NDVI (Normalized Difference Vegetation Index) ed EVI (Enhanced Vegetation Index), che stimano la qualità della vegetazione (in scala percentuale) sulla base della firma spettrale della vegetazione a terra.

(3) Anche la terza tipologia di Monitor-node risiede nel Cloud: si tratta di un ulteriore servizio esterno, che fornisce dati relativi alle previsioni meteorologiche entro 7 giorni, in funzione di una coordinata di riferimento fornita. Le condizioni meteo fanno più in generale riferimento alla città o alla località in cui giacciono i terreni coltivati.

- Analisi e Pianificazione

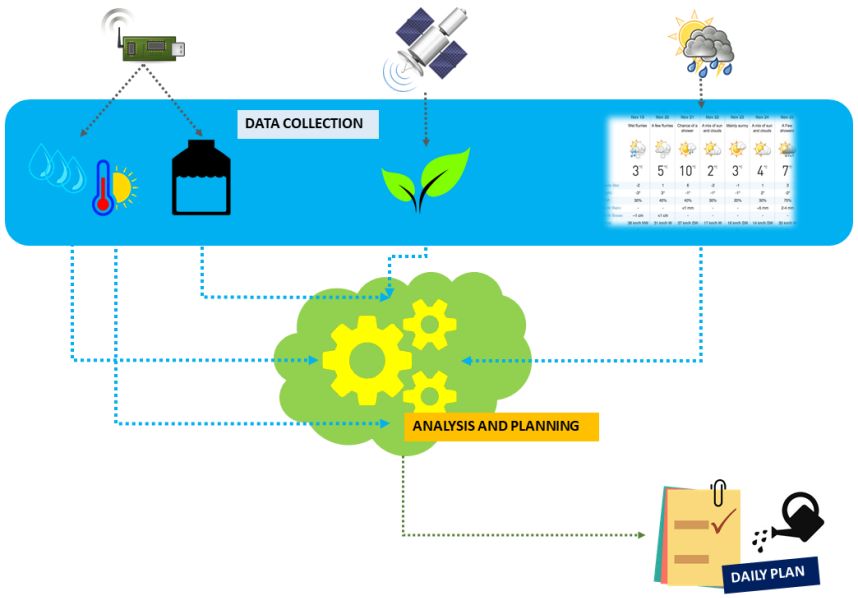

Fig. 3. Schema della fase di analisi e pianificazione, presso il modulo di Precision Farming nel Cloud.

Le fasi di analisi e pianificazione vengono eseguite su uno stesso modulo, che si prevede istanziato su nodo Cloud.
Il contenuto del piano giornaliero (o *Daily Plan*) indica, in modo specifico per ogni poligono, la quantità di acqua utilizzabile per ogni irrigazione nell'arco della giornata (sono previste 3 irrigazioni per ogni poligono).
Si descrive schematicamente la dinamica di genesi del piano giornaliero:
– Il modulo responsabile del reperimento del piano giornaliero inoltra la richiesta verso il modulo di analisi e pianificazione, allegando i valori medi di temperatura e umidità del

suolo registrati da ogni sensore, così come l'attuale quantità di acqua rimasta nel container, e la data prevista per il refill.
– Il modulo di analisi e pianificazione raccoglie (in aggiunta a quanto ricevuto dai dati sensoristici) le informazioni di qualità del suolo per i poligoni di riferimento, e le previsioni meteorologiche entro i 7 giorni che seguono l'istante della richiesta.
– L'insieme di questi dati, spefici per ognuno dei poligoni, viene analizzato, quindi associato a sintomi come secchezza del terreno, bassa qualità delle foglie, previsione di siccità o di inondazioni, e così via.
In base alle valutazioni congiunte, viene infine valutato il valore di un coefficiente nel range Reale [0.0, 1.0], che andrà a moltiplicare il valore della razione giornaliera di acqua, precedentemente valutata come:

$$(VolumeContainer/RemainingDays) * PolygonAreaCoverage \quad (1)$$

In questo modo, si garantisce che la quantità d'acqua giornaliera non ecceda mai rispetto al previsto, e che al più ne venga risparmiata una percentuale, con conseguente ottimizzazione economica.

- Esecuzione

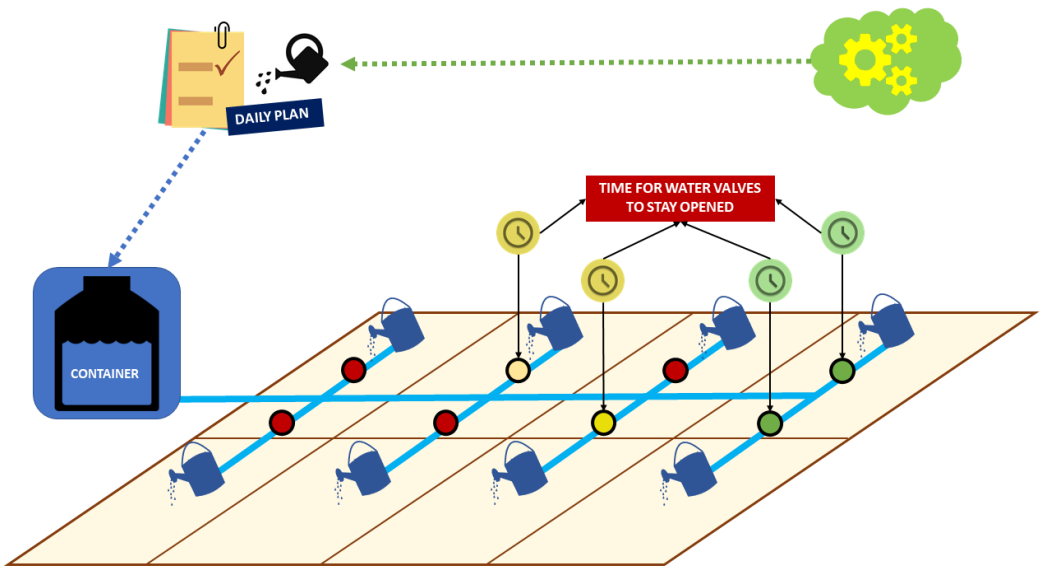

Fig. 4. Schema della fase di esecuzione. Applicazione del *Daily Plan* in termini di tempo di irrigazione, per ogni poligono

La fase di esecuzione inizia con la ricezione del *Daily Plan*.
Per ogni poligono, viene indicata la quantità di acqua destinata ad ognuna delle tre irrigazioni giornaliere.
Il modulo addetto al reperimento del piano ha l'incarico di informare i moduli esecutori, affinché vengano aperte/chiuse le valvole del sistema idrico: chiaramente, i moduli di esecuzione verranno eseguiti sui corrispondenti sensori IoT.

2.2 Architettura di sistema

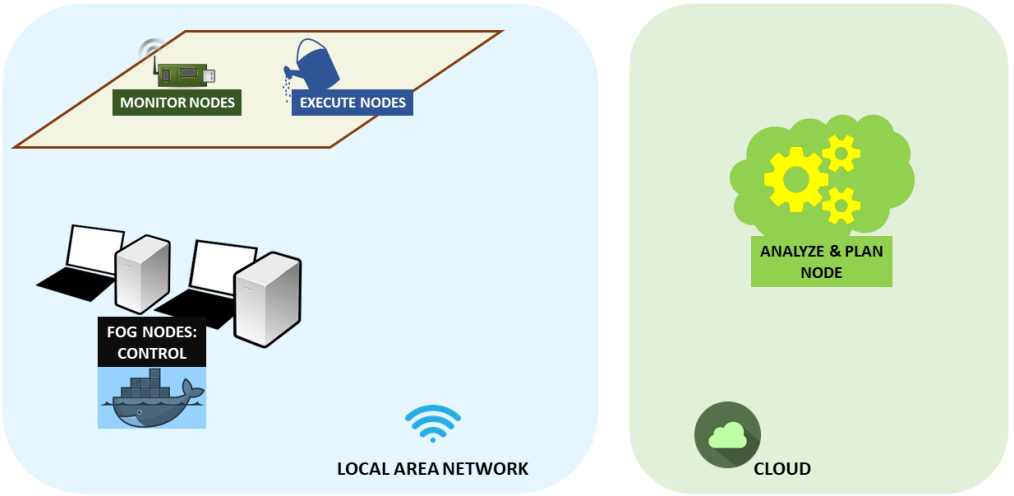

Fig. 5. Schema dell'istanziazione del software su nodi fisici.

Dopo aver introdotto l'organizzazione logica e l'interazione tra i vari componenti del sistema, si delineano ora le modalità per l'istanziazione dell'architettura software sui nodi del sistema distribuito.

- Per quanto riguarda i moduli di monitoraggio, questi verranno collocati su nodi fisici distinti, in corrispondenza dei sensori IoT utilizzati nei campi coltivati per l'acquisizione dei parametri di temperatura, umidità e immagini del suolo.
  In fase di sviluppo, tuttavia, tali moduli sono stati eseguiti sull'host locale, facendo quindi uso di un unico nodo fisico.
- Il set di micro-servizi, costituente il core dell'applicazione, designa un centro di controllo che provvede al soddisfacimento delle seguenti funzionalità:
  - Comunicazione con i moduli di monitoraggio;
  - Comunicazione con il modulo di analisi e pianificazione;
  - Comunicazione con i moduli di esecuzione;
  - Salvataggio di dati giornalieri all'interno di database locale;
  - Esposizione di un'interfaccia utente per il controllo e la gestione del sistema complessivo;
  - Backup di dati storici all'interno di un Cloud Storage;
  - Servizio di *alert* via e-mail, per notificare problemi di comunicazione con i device, o guasti degli stessi.

  La tecnologia dei Container permette di trattare questi moduli come micro-servizi isolati, che possono essere gestiti e posti in comunicazione dall'apposito tool di orchestrazione (i dettagli a riguardo verranno esposti nella sezione dell'articolo dedicata all'implementazione). Questo cluster di servizi corrisponde interamente al lato Fog dell'applicazione, ed è pertanto posizionata nelle vicinanze dell'utente:

Per quanto detto, quest'ultimo potrà potenzialmente essere distribuito su un singolo o su molteplici nodi fisici.
Ad ogni modo, non disponendo di un cluster adibito al Fog Computing, in fase di sviluppo si è scelto di istanziare il cluster sull'host locale.

- Come anticipato nell'analisi dei requisiti, il modulo responsabile dell'analisi e della pianificazione verrà istanziato su un nodo Cloud. Ulteriori indicazioni sulla piattaforma utilizzata verranno esposte nella sezione dell'articolo dedicata all'implementazione.

3 IMPLEMENTAZIONE

In questa sezione si descrivono i dettagli relativi all'implementazione dell'applicativo, tra cui linguaggi, frameworks e librerie utilizzate nel codice.


3.1 Linguaggi di programmazione e principali librerie

Il linguaggio scelto per lo sviluppo dell'applicazione è *Python*. In particolare, si è fatto utilizzo del micro-framework web *Flask* come piattaforma software di supporto allo sviluppo dell'applicativo.

La scelta è dettata dal fatto che Flask supporta la comunicazione stateless: in particolare, trattandosi di un'applicazione che segue lo stile architetturale *RESTful (Representational State Transfer)*, l'utilizzo di questo framework si è rivelato particolarmente adeguato ed ha permesso di produrre codice uniforme e di facile lettura. Per inoltrare richieste HTTP, alla base delle REST API, si è fatto uso della API Python *Requests*.

L'aver utilizzato il linguaggio Python non è tuttavia una scelta esclusiva: è possibile di fatto scrivere servizi utilizzando linguaggi disparati, purché supportino invio e/o ricezione di richieste HTTP.

3.2 Docker Containers

Per il packaging dei singoli moduli (con annesso ambiente di esecuzione e librerie), sia all'interno del cluster fog che per il modulo in esecuzione sulla piattaforma Cloud, è stato utilizzato Docker. I singoli container utilizzano l'immagine *Ubuntu 20.04*, oltre alla quale vengono installati tutti gli strumenti necessari per lo sviluppo in Python (*python3*, *python3-dev*, *pip*, ...).

3.3 AWS: *Elastic Beanstalk* e *S3*

Trattandosi di un servizio web, per la distribuzione del modulo *Precision Farming* è stato utilizzato il servizio Elastic Beanstalk offerto da Amazon AWS. Il sistema offre, in modo automatizzato ed intuitivo, provisioning delle risorse, bilanciamento del carico, dimensionamento automatico e monitoraggio. Amazon S3, invece, viene utilizzato per l'archiviazione dei dump del database dei singoli cluster fog.

3.4 *Kubernetes* e *Minikube* per l'orchestrazione dei containers

Kubernetes è stato utilizzato come sistema open-source per l'orchestrazione dei container. La scelta è dettata dalle funzionalità offerte dal sistema come il deployment automatico, la scalabilità e la gestione dei guasti. Per lo sviluppo dell'applicazione è stato scelto *Minikube*, progetto open-source che permette l'esecuzione in locale di un nodo Kubernetes: ad ogni modo, l'utilizzo di *Minikube*

è servito principalmente per semplicità di implementazione, ma non è preclusa la possibilità di effettuare il deployment dell'applicazione su un ambiente Kubernetes puro.

3.5 I moduli IoT

Andiamo a trattare le funzionalità implementate nei moduli IoT.

- **Client SSDP:** per automatizzare l'esecuzione dei nodi IoT, questi inviano in broadcast un messaggio *SSDP* a cui un *proxy server* risponderà comunicando il proprio *indirizzo IP*. Conoscere l'indirizzo dei *proxy server* è necessario in quanto i client non comunicano direttamente con il cluster fog, a causa delle limitazioni poste da Minikube di cui tratteremo più avanti. In caso di errori nella comunicazione con un *proxy server*, viene avviato nuovamente il client SSDP per riottenere l'indirizzo IP di un proxy valido.
- **Registrazione del dispositivo:** viene inviata al *proxy server* la richiesta di registrazione del dispositivo all'interno della base di dati mantenuta nel cluster fog.
- **Invio delle letture (per i soli nodi di monitoraggio):** i nodi di tipo *monitor* inviano, ad intervalli di tempo regolari, delle richieste di registrazione delle proprie letture ad un *proxy server*. Il parametro relativo all'intervallo delle letture viene impostato inizialmente dal file *'config.json'*, ma può essere modificato a runtime dalla dashboard a disposizione dell'utente.
- **Ricezione delle informazioni sul planning (per i soli nodi di esecuzione):** ad intervalli di tempo regolari, i dispositivi dediti all'esecuzione ricevono dal cluster informazioni riguardo alla quantità di acqua, in m^3, di acqua che deve essere inoltrata verso il campo coltivato. In base alla velocità di fuoriuscita dell'acqua dalle tubature, viene calcolato il tempo di apertura della valvola meccanica.
- **Controllo dello stato:** ad intervalli di tempo regolari, i dispositivi vengono contattati dal cluster per verificarne lo stato di vita. A tale fine è stato utilizzato il micro-framework Flask per la ricezione delle richieste GET.

3.6 Il modulo di Leaf Disease Detection

Per quanto rientri nella definizione di nodo IoT, questo modulo supporta delle funzionalità più articolate, come spiegato a seguire. Il modulo di Leaf Disease Detection, scritto in linguaggio Python, ha lo scopo di analizzare immagini (Image Processing) fornite dai sensori IoT per valutare il grado di deterioramento delle piantagioni. Il funzionamento del modulo è riportato di seguito:

- **Cattura immagine:** Il sensore IoT viene sollecitato, a cadenza regolare (configurabile in fase di inizializzazione), per la cattura di immagini direttamente nell'area vegetata d'interesse.
- **Analisi dell'immagine:** Attraverso l'utilizzo della libreria OpenCV, che applica vari filtri all'immagine, è possibile individuare, sfruttando un range di colori (facendo uso della codifica L.A.B. e HSV), delle zone nella coltivazione in cui è in corso un infezione (ad esempio causata da funghi), oppure uno stato di disidratazione della piantagione. Per ottenere un'analisi più precisa è necessario mantenere il riferimento di un'immagine della piantagione in uno stato di salute. L'analisi, in base anche ad un confronto con l'immagine di riferimento, restituisce la foto fornita in input dove vengono però evidenziate le criticità, ed un valore che specifica la percentuale malata della pianta rispetto alla sua totalità. In base a questo valore viene deciso (basandosi su delle soglie predefinite) se inviare una email di notifica all'utente, allegando la foto della situazione della piantagione, oppure no.
- **Risposta alla dashboard:** Nella dashboard, in caso di media o alta criticità, verrà riportato un messaggio di avvertenza, relativamente allo stato attuale dell'area monitorata.

3.7 Il Proxy Server

Una delle limitazioni riscontrate a causa dell'utilizzo di *Minikube* è l'impossibilità di raggiungere il cluster dall'interno della rete locale, se non dal computer su cui l'applicativo è in esecuzione. Per questo motivo è stato implementato un semplice *proxy server* che riceve le richieste dei nodi IoT e le inoltra al cluster. L'implementazione del Proxy Server è stateless. Questo permette di avviarne più repliche, per avere una maggiore tolleranza ai guasti. Seguono le altre funzionalità del *Proxy Server*.

- **Server SSDP:** processa i messaggi in arrivo dai nodi IoT fornendo il proprio indirizzo IP.
- **Ottenere l'indirizzo IP esposto dal fog cluster:** tramite il metodo *getServiceExternalIP* del modulo **minikubeservice.py**, ottiene l'indirizzo IP esposto dal servizio *collect data*, presente all'interno del cluster fog. Per tollerare errori nella comunicazione, il metodo viene richiamato finché il servizio non torna disponibile.

3.8 Fog cluster

I servizi offerti da Kubernetes possono essere schematizzati, a grandi linee, come segue:

- Deployments
- Services
- Jobs
- Cron Jobs
- Persistent Volumes
- Horizontal Pod Autoscaler (HPA)

Di questi, quelli utilizzati nel progetto, sono dichiarati attraverso i file di configurazione *yaml*, disponibili nella cartella *Server/yaml*. Si procede a descrivere le funzionalità dei servizi implementati.

- **Deployment:** possono essere considerati come la singola unità applicativa in esecuzione continua all'interno del cluster.
  - **MySQL:** istanza del database in esecuzione all'interno del cluster fog.
  - **Collect Data:** microservizio che si occupa della ricezione delle richieste di registrazione dei dispositivi e del raccoglimento delle loro letture, andandole ad inserire all'interno del database locale. Viene contattato direttamente dal *proxy server*.
  - **Dashboard:** microservizio che si occupa della gestione della dashboard consultabile dall'utente finale tramite interfaccia web. Le funzionalità offerte dalla dashboard sono:
    * Visualizzazione delle previsioni meteo entro 7 giorni.
    * Gestione dei dispositivi (controllo dello stato, modifica del poligono di appartenenza, impostazione della periodicità delle letture, eliminazione dei dispositivi)
    * Consultazione dello storico dei dump del database da S3
    * Gestione dei poligoni (inserimento, eliminazione)
    * Gestione dello stato di salute di una piantagione (controllata tramite sensore con fotocamera)
  - **DB Connector:** microservizio che si occupa della gestione delle operazioni *CRUD* richieste dalla *dashboard*.
  - **Send E-mail:** microservizio che si occupa dell'invio di e-mail di avviso nel caso in cui un dispositivo dovesse risultare non raggiungibile, oppure per segnalare una criticità nello stato di salute di una piantagione.
- **Service:** rendono possibile la comunicazione tra i diversi servizi del cluster. Possono essere di due tipi: **NodePort**, per rendere raggiungibile un deployment solamente dall'interno del cluster, **LoadBalancer**, per rendere raggiungibile un deployment anche dall'esterno

del cluster. Nel secondo caso, *Kubernetes* indicherà l'indirizzo IP esterno da contattare per raggiungere il deployment desiderato (*i.e. dashboard*).

- **Job:** permettono di indicare delle operazioni che devono essere eseguite un'unica volta all'avvio del cluster. Per esempio, è stato creato il *job instantiate database* per la creazione ed il popolamento della base di dati locale.
- **CronJob:** a differenza dei job, i cronjob vengono eseguiti ad intervalli regolari di tempo.
  - **Calculate value:** si occupa di aggregare le letture dei dispositivi di monitoraggio presenti all'interno della base di dati e contattare il modulo *Precision Farming* per ottenere informazioni sul planning giornaliero. Ricevuta risposta, la memorizza all'interno di un file *.json* e, per ogni gruppo, contatta i singoli dispositivi di esecuzione per indicare la quantità di acqua (in m^3) che deve essere offerta al campo coltivato.
  - **Check device status:** si occupa di contattare i dispositivi registrati all'interno della base di dati locale per verificarne lo stato di vita tramite richieste *GET*. Nel caso di non risposta, contatta il servizio *Send E-mail* ed aggiorna lo stato del dispositivo come non disponibile.
  - **DB Dump:** si occupa del backup del database, con la creazione di un file di dump, e dell'eliminazione delle letture degli ultimi x giorni, dove x è un parametro configurabile.
  - **Dump upload:** si occupa dell'upload su AWS S3 del dump creato dal precedente cronjob.
- **PersistentVolume e PersistentVolumeClaim:** rendono possibile la creazione di uno spazio persistente, condiviso tra le varie entità del cluster. All'interno del programma vengono usati da:
  - **MySQL:** la persistenza è necessaria per mantenere lo stato del database anche dopo un riavvio del cluster.
  - **DB Dump-Dump upload:** la persistenza condivisa è necessaria per permettere a *Dump upload* di accedere all'ultimo dump del database effettuato.
  - **Calculate value-Dashboard:** tra le informazioni ottenute con il planning, sono presenti anche quelle relative alla qualità della vegetazione di ogni singolo campo coltivato. Queste informazioni vengono utilizzate dalla dashboard, che accede al file *.json* memorizzato dal modulo *Calculate value*, per mostrarle all'utente finale.
- **Secret:** implementa un meccanismo di sicurezza che evita il reperimento delle password via codice. Questo è possibile grazie all'impostazione di variabili d'ambiente, a cui è possibile accedere solamente a run-time. Viene utilizzato per lo storage della password per l'accesso al database.
- **Horizontal Pod Autoscaler:** permette di effettuare l'auto-scaling in orizzontale di un deployment in riferimento a dei parametri, potendo impostare il numero minimo e massimo di repliche in esecuzione contemporaneamente. Questo servizio si rivela utile nel caso in cui si ha a disposizione un cluster fog distribuito su più nodi. Per il programma, l'HPA è stato applicato al deployment *Collect data*, con numero minimo e massimo di repliche rispettivamente 2 e 10. Il parametro secondo cui effettuare lo scaling in orizzontale è l'utilizzo della CPU, impostato al 50

3.9 Il modulo di Precision Farming

Nel modulo di Precision Farming prendono atto le fasi MAPE di Analisi e Pianificazione, a partire dai dati reperiti nella fase di Monitoraggio.

Come anticipato, questo modulo Python viene eseguito sulla piattaforma Cloud Amazon Elastic Beanstalk.

Si descrivono i punti focali dell'implementazione:

- **I servizi OpenWeather ed Agro API per il retrival dei dati di monitoraggio:** per il retrival dei dati di monitoraggio non relativi alla sensoristica impiantata in loco, si è fatto uso delle API esposte dai siti *OpenWeather [3]* (per le previsioni meteorologiche) e *Agro API* [1] (per i dati dalle immagini satellitari Landsat 8 e Sentinel 2). L'accesso ai servizi è garantito attraverso HTTP requests, in linea con lo stile RESTful per la comunicazione tra le componenti del software.
  Per l'accesso ai servizi, è necessario in primo luogo creare un account gratuito ed ottenere una *APIkey*, che dovrà essere inserita come parametro nelle richieste inoltrate ai suddetti siti. Per ricevere i dati relativi alle specifiche aree coltivate, vengono generate delle *Polygon entities* (attraverso apposita API) fornendo le coordinate geografiche dei vertici di queste.
- **Risposta al Fog cluster:** Il Fog cluster (tramite il CronJob *Calculate Value*) inoltra periodicamente la richiesta di pianificazione verso il modulo di Precision Farming.
  Effettuato il retrival di tutti i dati di monitor necessari, questi ultimi vengono utilizzati per popolare oggetti appartenenti a classi Python adeguatamente progettate ed implementate. A tal proposito, e per garantire una visione globale delle operazioni eseguite in questa fase MAPE, si allega un *UML Class Diagram d'implementazione*, presentando le entità coinvolte, i principali attributi e i metodi implementati, nonché le relazioni che sussistono tra le istanze delle classi:

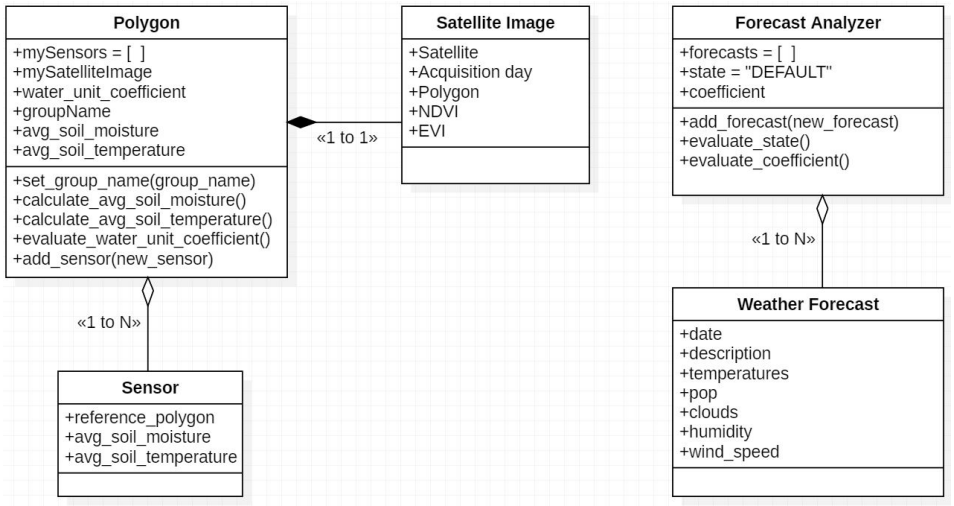

Fig. 6. UML Class Diagram: entità coinvolte nel modulo di analisi e pianificazione.

Terminate le fasi di Analisi dei dati e della conseguente Pianificazione della gestione giornaliera delle risorse idriche, tutti i parametri necessari all'esecuzione del piano vengono ritornati in risposta al Fog cluster, in formato json.

[1] https://agromonitoring.com/

4 TESTING

4.1 Cluster Fog

Per il testing del *cluster fog*, disponibile nella cartella *Test/Fog Cluster*, l'idea è stata quella di iniettare artificialmente dei guasti all'interno dell'istanza di Kubernetes: questo è stato possibile andando ad ottenere, per ogni singolo deployment, la lista dei container Docker in esecuzione, ed eliminando da essa uno dei pod tramite il segnale di *kill*.

Successivamente, si va a controllare che il deployment (dopo un periodo di tempo che aumenta in modo esponenziale in base al numero di kill andate a segno) sia tornato up and running.

Dai test, si evince che Kubernetes fornisce effettivamente un supporto dinamico alla tolleranza ai guasti.

4.2 Precision Farming

Per quanto riguarda il testing del modulo di pianificazione, agente su piattaforma cloud *AWS Elastic Beanstalk*, si deve precisare che sono state applicate strette restrizioni all'utilizzo intensivo dei servizi esterni (per il reperimento dei dati meteorologici e satellitari).

Nonostante ciò, si ritengono comunque esaustivi i casi di test realizzati, trattandosi di un modulo che non viene mai posto sotto particolare pressione (ogni fog cluster sottopone al più 1 richiesta al giorno).

A seguire, si presentano i risultati ottenuti effettuando test prestazionali sul tempo di riposta alla richiesta di pianificazione giornaliera: il tempo medio di risposta viene calcolato su 100 misurazioni, utilizzando gli approcci sotto elencati.

- single-thread: 100 chiamate serializzate.
- multi-thread: 10 threads, 10 chiamate serializzate per ciascuno.

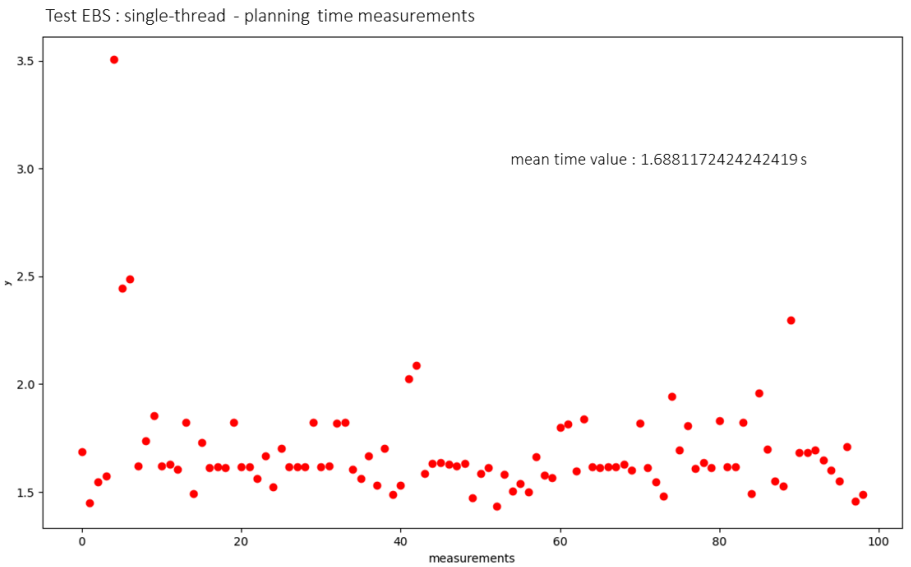

Fig. 7. Istanza di test, prestazioni in tempi di risposta nel caso di chiamata a singolo thread.

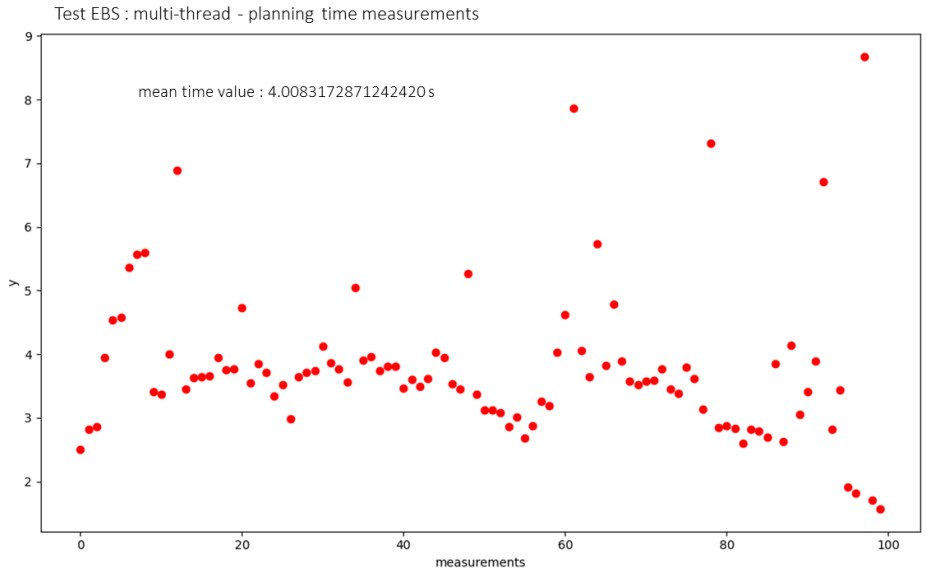

Fig. 8. Istanza di test, prestazioni in tempi di risposta nel caso di chiamate in concorrenza (10 threads).

Dai test effettuati si nota che i tempi di risposta aumentano all'aumentare delle richieste in parallelo. Per quanto possa apparire sintomo di non scalabilità dell'applicativo, si tratta di fatto di un problema di soglia: è possibile infatti impostare sulla piattaforma le modalità di scaling-out e le soglie entro cui effettuarlo.

5 SVILUPPI FUTURI E LIMITAZIONI RISCONTRATE

La vera potenzialità di *Foggy Day* risiede nella possibilità di essere facilmente estesa e/o migliorata: l'architettura a microservizi e la comunicazione REST tra i building bocks permettono infatti uno sviluppo continuo, semplice ed intuitivo, utilizzando i linguaggi più disparati.

Il rilascio di nuove versioni, tra l'altro, non soffrirebbe l'integrazione di componenti legacy, grazie all'utilizzo dei Docker containers, che permettono la configurazione dell'ambiente d'esecuzione *a piacimento* del programmatore, in modo isolato e indipendente dagli altri servizi.

Proprio per questi motivi, non mancano idee per sviluppi futuri dell'applicativo, e in questo paragrafo si vanno ad delineare quelle salienti.

5.1 Rete Neurale a supporto del modulo di Precision Farming

L'attuale realizzazione del piano giornaliero di gestione delle risorse idriche prevede l'implementazione di un *modello diretto*, implementato nel modulo di Precision Farming, dove i parametri di input sono soggetti ad elaborazioni in base a principi fisici, e viene così dedotto un risultato finale.

Applicare un *modello diretto* che sia in grado di elaborare una risposta corretta ed accurata (rispetto alle condizioni reali correnti) risulta un'impresa ardua, per il semplice fatto che gli eventi fisici reali spesso e volentieri non possono esser ricondotti ai *modelli fisici* noti, dipendendo di fatto da condizioni al contorno in continua variazione, e da parametri non sempre misurabili.

Ciò che può intervenire in aiuto, a tal proposito, è l'intelligenza artificiale, e l'adozione di ciò che è noto come *modello inverso*.

L'idea è quella di addestrare una *rete neurale*, che sia in grado di stabilire, in funzione dei dati di monitoraggio (utilizzati come parametri di input alla rete stessa), quale sia effettivamente la quantità d'acqua da utilizzare per le irrigazioni: una rete neurale di tale portata necessiterebbe ovviamente di una o più epoche di addestramento, per le quali sarebbe possibile utilizzare i dati raccolti dall'applicazione, in corrispondenza agli output reali misurabili sul campo.

Una volta addestrata, tale rete neurale potrebbe essere interamente sostituita all'attuale modulo di Precision Farming, massimizzando l'efficienza dell'applicazione.

5.2 Rete Neurale a supporto del modulo di Leaf Disease Detection

L'implementazione attuale del modulo di Leaf Disease Detection sfrutta unicamente come modello di analisi la tecnica dell'Image Processing. Questo (per vari motivi, come le condizioni delle luci, e la risoluzione della macchina) fornisce un risultato non perfettamente accurato.

Un'idea vincente potrebbe esser quella di utilizzare una rete neurale, nello specifico di tipo *CNN (Convolutional Neural Networks)* largamente utilizzata in ambito di Image recognition.

Tramite un periodo di training queste reti neurali saranno in grado di rilevare precocemente la malattia di una pianta, categorizzandola in modo da poter intervenire per tempo prima che i sintomi manifestati distruggano irrimediabilmente la pianta stessa.

Un'importante limitazione riscontrata nell'implementazione del modulo di Leaf Disease Detection riguarda la difficoltà nel categorizzare le differenti specie di piante sulla base della propria colorazione: non trattandosi di un parametro fisico esatto, è pressoché impossibile ottenere valutazioni accurate, soprattutto all'occorrenza di numerose varietà agricole (quindi una gamma molto estesa di colori).

L'utilizzo dell'intelligenza artificiale può venire in aiuto anche in questo contesto, poiché una rete neurale addestrata adeguatamente sarà in grado di fornire risultati pressoché esatti al variare delle istanze reali di specie vegetali.

5.3 MAPE gerarchico: da *Master-Worker* a *Regional* style

L'attuale architettura di Foggy Day rispecchia una struttura MAPE gerarchica, di tipo *Master-Worker*: un singolo nodo Master, con responsabilità di Analisi e Pianificazione, si interfaccia a numerosi nodi di monitoraggio, distribuiti in uno o più territori distinti ma pressoché vicini tra loro, e prende decisioni a livello globale da inoltrare ai nodi d'esecuzione (fisicamente coincidenti o prossimi ai nodi di monitoraggio).

Un idea per incrementare la scalabilità dell'applicazione è quella di trasformare questa architettura nel pattern MAPE gerarchico *Regional*. L'idea è di creare un nodo Master per ogni cluster fog: in questo modo, ogni singolo fog cluster farà riferimento ad un nodo *privato* per la fase di analisi, e questi ultimi inoltreranno le informazioni *regionali* al nodo di pianificazione, che, in base alla visione di insieme, restituirà un planning globale per ottimizzare le risorse idriche, geograficamente distribuite in regioni dislocate.

Questo tipo di architettura si presta a situazioni in cui uno stesso utente possegga coltivazioni geograficamente distribuite, ed abbia necessità di ottimizzare risorse agricole condivise in tutte le regioni geografiche.

5.4 Implementazione reale dei sensori monitor ed execute

Al momento i sensori IoT sono simulati in locale tramite script in *Python*. Sono state effettuate delle prove andando ad utilizzare un *Raspberry PI*, su cui è stato eseguito lo script in *Python*, e non sono state riscontrate criticità di alcun tipo.

La rilevazione della temperatura e dell'umidità, per quanto riguarda i sensori di tipo *monitor*, avviene tramite l'utilizzo di una libreria *dummy* che va a generare dei numeri random in un

intervallo definito. L'implementazione reale, dopo alcune ricerche sul web, non richiederebbe un grande esborso economico [4].

Anche per i sensori di tipo *execute*, sono attualmente disponibili sul mercato diverse soluzioni per l'implementazione reale dell'idea alla base dell'applicativo. Sono disponibili sul mercato valvole meccaniche che è possibile controllare da remoto [5], oltre a librerie software per l'implementazione in diversi linguaggi di programmazione [7].

5.5 Limitazioni nell'inizializzazione lato utente

Nella fase di inizializzazione dell'applicativo, è importante specificare che ci si trova davanti ad una limitazione, perlopiù sperimentata in fase di sviluppo, ma che dovrebbe poter essere superata una volta giunti in fase di distribuzione.

Si tratta di una limitazione imposta dal server esterno, riguardante il tracciamento dei poligoni che dovranno corrispondere alle aree coltivate: il server, infatti, utilizza un *tool grafico [6]* online per il tracciamento manuale dei poligoni su mappe geo-referenziate. Una volta specificate le aree di interesse, il tool grafico fornisce le coordinate geografiche dei vertici di ogni poligono, da utilizzare nelle chiamate specifiche al server.

Questa limitazione è chiaramente superabile non appena si disponga di sensori dotati di GPS: in tal caso, sarà sufficiente piazzare unità sui vertici dei campi coltivati, in modo da informare autonomamente riguardo alle coordinate geografiche d'interesse.

5.6 Limitazioni Minikube: proxy server e singolo nodo

Come già visto nella sezione riguardante *l'implementazione*, Minikube non permette l'ingresso diretto al cluster Kubernetes dall'interno della rete locale. Questo ha reso necessario l'implementazione di un *proxy* per l'inoltro delle richieste provenienti dai sensori installati. Altra limitazione riscontrata è l'impossibilità di eseguire il cluster con più nodi, non potendo dunque sfruttare a pieno le potenzialità offerte da Kubernetes per quanto riguarda distribuzione e tolleranza ai guasti.

Come già detto il programma è stato sviluppato utilizzando i servizi messi a disposizione da Kubernetes. E' stato possibile vedere come, utilizzando **K3S** (https://k3s.io/), altro software open-source che permette di eseguire un'istanza di Kubernetes in locale, le due limitazioni non sono presenti. E' dunque possibile affermare che le due limitazioni riscontrate riguardano l'ambiente di sviluppo utilizzato, ma non l'applicativo in se.

REFERENCES

[1] NASA, United States Geological Survey (USGS), web site: *https://www.usgs.gov/core-science-systems/nli/landsat/* .
[2] European Space Agency (ESA), progetto Copernicus, web site: *http://www.esa.int/Applications/Observing_the_Earth/Copernicus/*.
[3] OpenWeather, web site: *https://openweathermap.org/*
[4] Sensore per Raspberry PI: *https://www.amazon.it/GeeekPi-Pressione-Temperatura-Illuminazione-Raspberry/dp/B07TZD8B61/*
[5] Valvola meccanica WiFi: *https://www.amazon.it/compatibile-telecomando-automazione-regolatore-Manipolatore/dp/B084VPD5TM/*
[6] Agromonitoring - Visual Tool, web site: *https://agromonitoring.com/create-polygon*
[7] Libreria Tuya: *https://github.com/clach04/python-tuya*
[8] Libreria OpenCV: *https://opencv.org/*