

GeoServer nel Cloud

Un caso di studio sulle modifiche architetturali
nel passaggio a piattaforme Cloud

Federico Cacco

Laurea Magistrale in Informatica

Università degli Studi di Padova
Dipartimento di Matematica

11 Ottobre 2013

Sommario

- 1 Scopo del progetto
- 2 Approccio architetturale per la risoluzione delle problematiche affrontate
- 3 Caso d'uso reale per verificare le soluzioni proposte
- 4 Ambiente Cloud utilizzato per implementare l'architettura proposta
- 5 Test effettuati
- 6 Conclusioni
- 7 Sviluppi futuri

Scopo del progetto

Le architetture delle piattaforme Cloud sono del tutto diverse da quelle convenzionali

- Applicazioni \Rightarrow Fornite come servizi web
- Istanza \Rightarrow Entità operativa che fruisce i servizi
- Elasticità \Rightarrow Capacità di adattarsi (*scaling*) all'esigenza corrente
 - Ottenuta mediante la replicazione delle istanze



Applicazioni progettate per piattaforme convenzionali sono inadeguate
agli ambienti Cloud

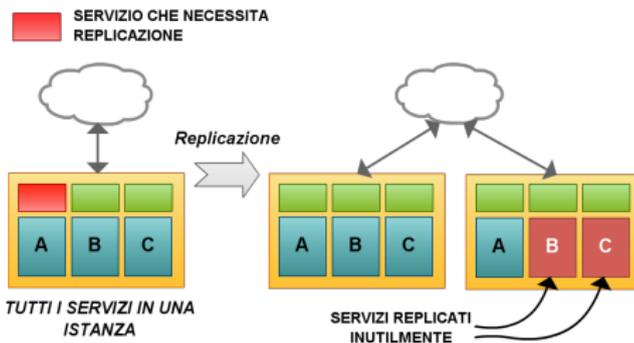
Quali considerazioni architetturali sono necessarie?

Scomposizione dei servizi

Necessario replicare solamente i componenti che ne hanno reale necessità

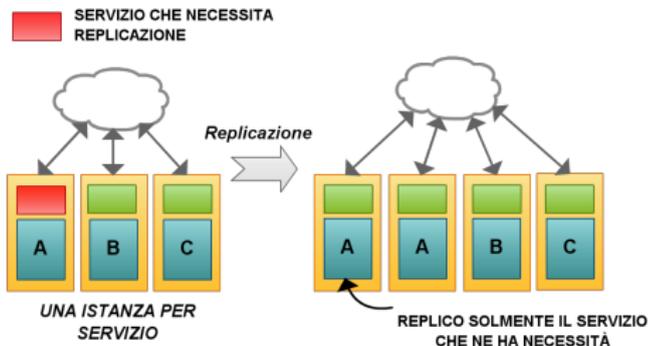
Tutti i servizi in una unica istanza

- Per scalare un servizio devo replicare l'istanza che fornisce tutti i servizi



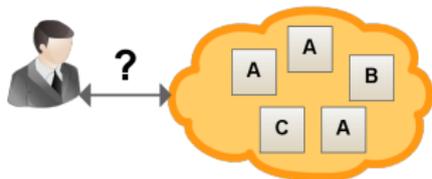
Una servizio per istanza

- Per scalare un servizio posso replicare solamente l'istanza che lo fornisce



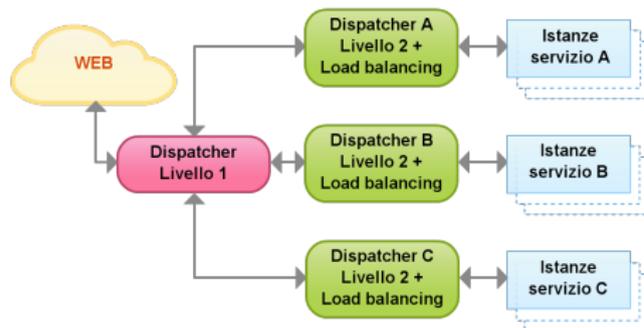
Rete di Dispatcher

La replicazione delle istanze che compongono il web server deve essere trasparente



- L'utente non deve preoccuparsi di inviare la richiesta all'istanza in quel momento libera
- Deve essere presente un unico punto d'accesso al web server
 - *Indipendentemente dal numero di repliche e dal numero di servizi forniti*

Soluzione: Rete di dispatcher



- Secondo livello
 - *Gestisce repliche di istanze che forniscono lo stesso servizio*
 - *Bilancia il carico di lavoro*
- Primo livello
 - *Instrada le chiamate al corretto dispatcher di secondo livello*
 - *Esponde un unico punto d'accesso al web server*

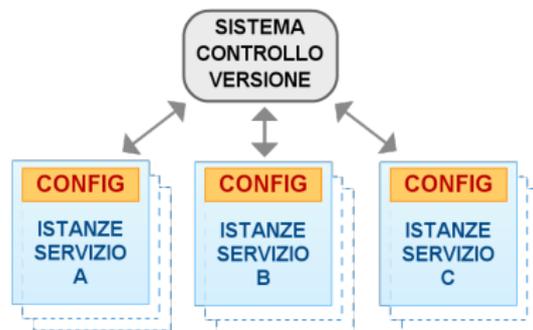
Configurazione comune delle istanze

- Tutte le istanze devono avere la medesima configurazione
- Modifiche alla configurazione di una istanza devono ripercuotersi su tutte le altre

Soluzione 1: Configurazione in uno spazio condiviso



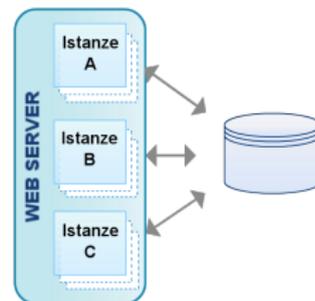
Soluzione 2: Configurazione locale sincronizzata



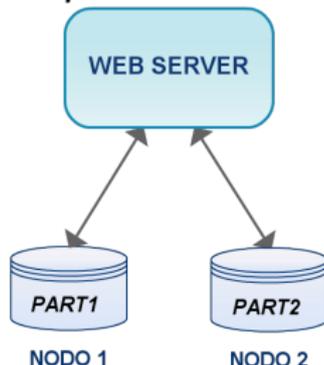
La scelta dipende dall'architettura dei servizi e dalla piattaforma Cloud utilizzata

Memorizzazione dei dati

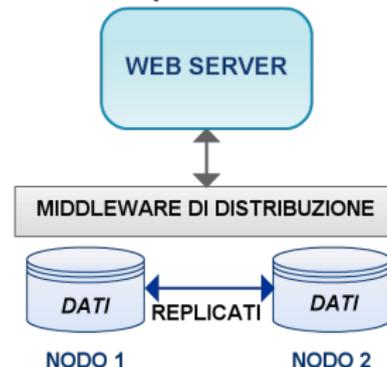
- Database soggetto a numerosi accessi simultanei
- Necessità di una sua distribuzione
 - *Partizionamento*
 - *Replicazione*



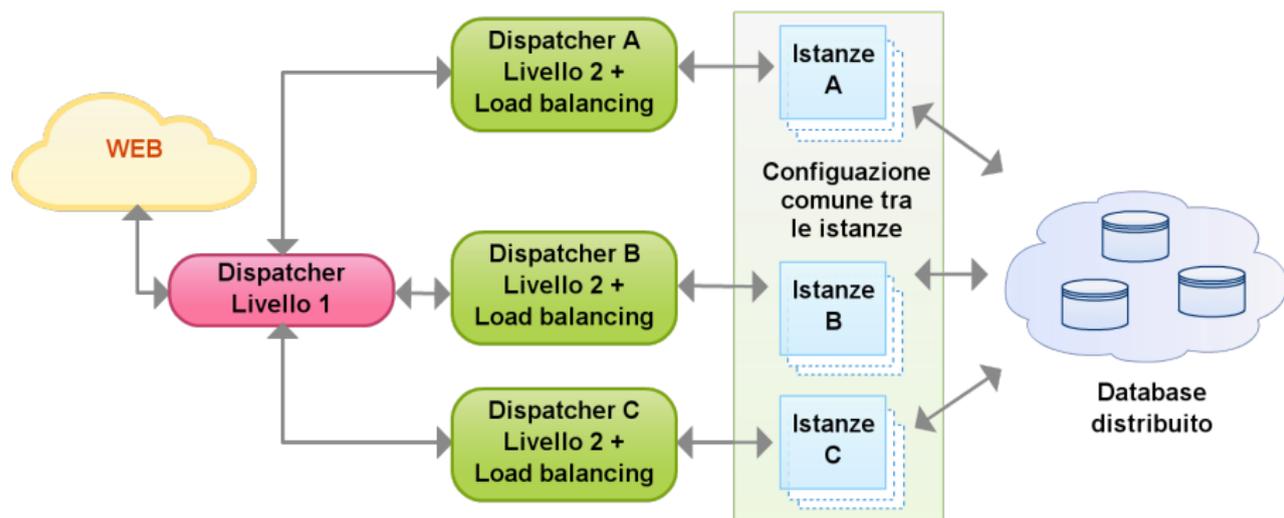
Distribuzione mediante partizionamento



Distribuzione mediante replicazione



Visione d'insieme

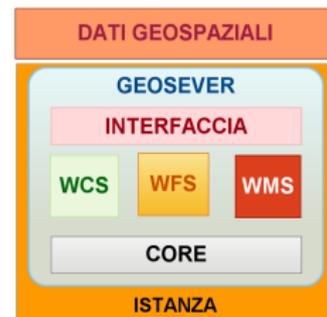


GeoServer

Caso di studio concreto: GeoServer

Server web geospaziale che fornisce i servizi

- *Web Coverage Service*
- *Web Feature Service*
- *Web Map Service*



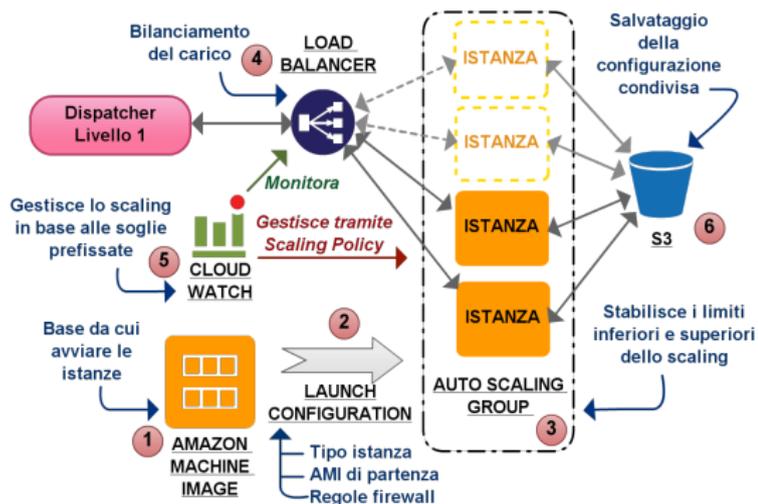
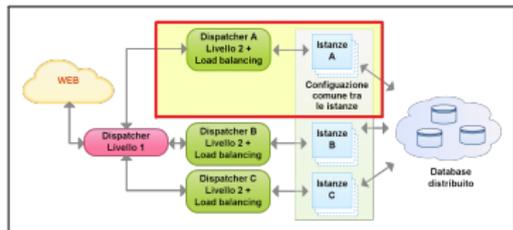
Servizi con differenti carichi computazionali

- *Scomposizione per poter avviare istanze contenenti i singoli servizi*



Strumenti AWS utilizzati

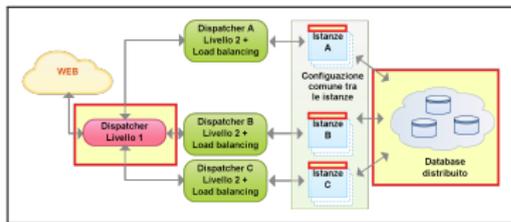
Ambiente Cloud utilizzato: Amazon Web Services



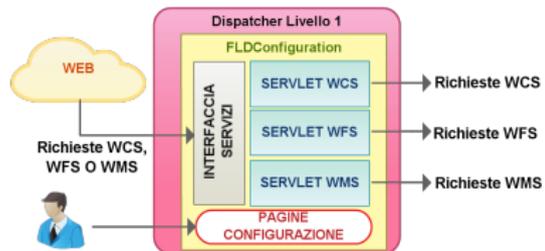
Metrica per lo *scaling* \Rightarrow Latenza

- Rilevata nei Load Balancer di secondo livello

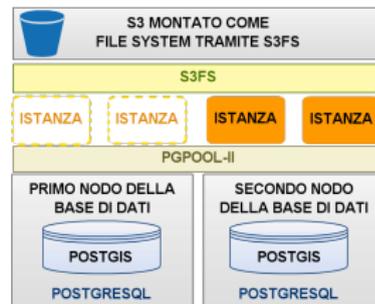
Strumenti esterni ad AWS utilizzati



- Dispatcher di primo livello

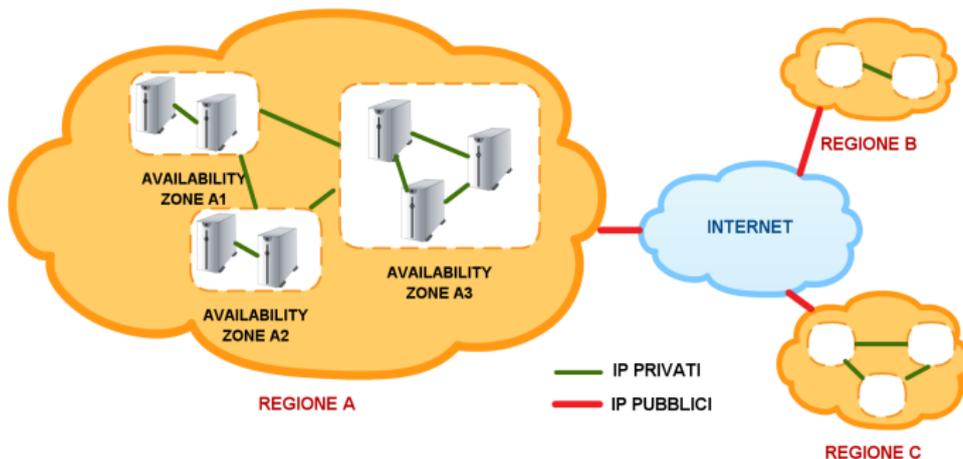


- S3FS (Middleware supporto S3)
- Pgpool-II (Middleware di distribuzione)
- PostgreSQL (Data Base Management System)
- PostGIS (Estensione per dati geospaziali)



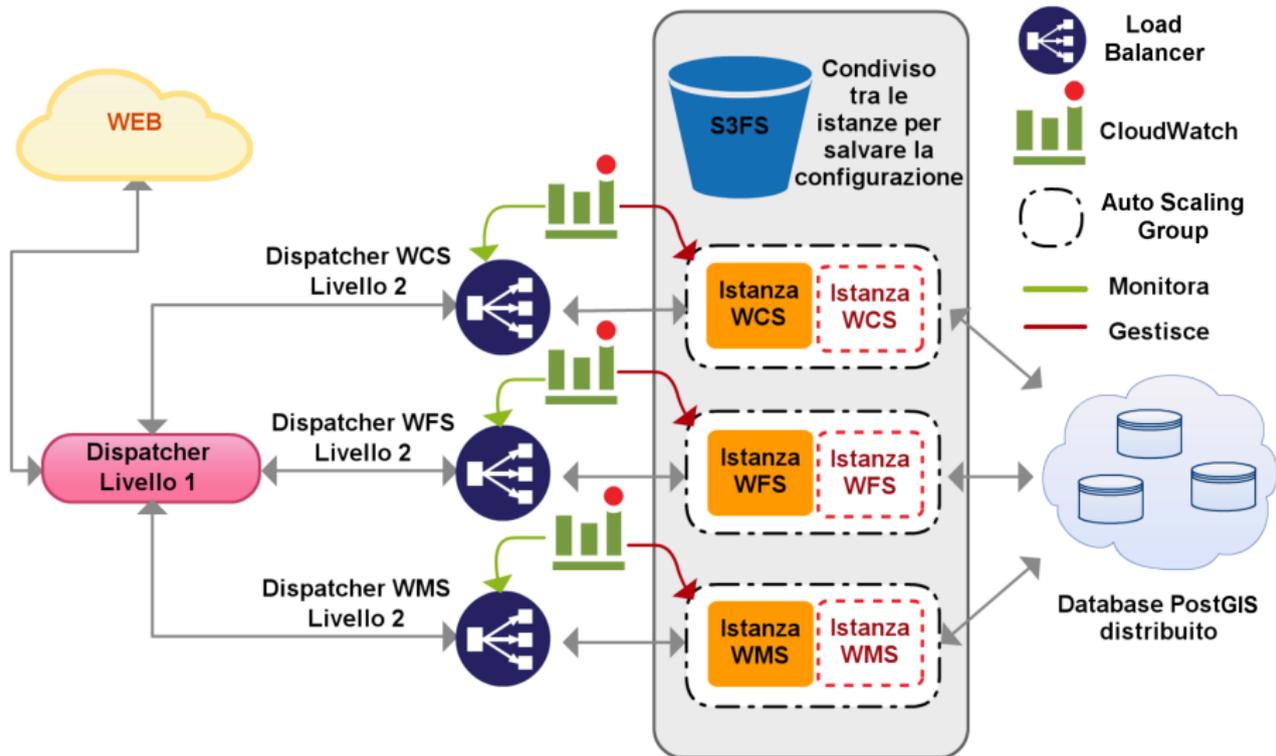
Regioni e Availability Zone

- Regione: Sedi su cui è collocato AWS nel mondo
- Availability Zone: Sotto-suddivisione delle Regioni



Le Availability Zone consentono di implementare architetture affidabili impedendo la diffusione dei guasti

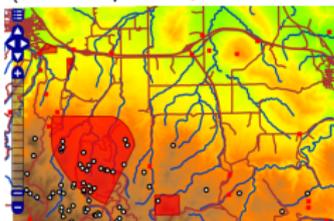
Visione d'insieme



Configurazione dei test

- Simulati utenti contemporanei che invocano i servizi
- 2 serie di dati geospaziali suddivisi in 2 workspace

sf (città di Spearfish, South Dakota)

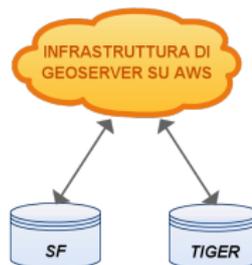


tiger (città di New York)

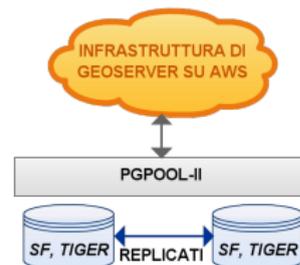


- 2 configurazioni di distribuzione della base di dati
 - *Partizionamento*
 - *Replicazione*

Distribuzione mediante partizionamento



Distribuzione mediante replicazione



Availability Zone

La distribuzione nelle Availability Zone per aumentare l'affidabilità introduce latenze?

- Simulati 5 utenti contemporanei

Richiesta	Diversa AZ		Stessa AZ	
	Lat. media (ms)	Throughput (rich/sec)	Lat. media (ms)	Throughput (rich/sec)
WMS_getMap (sf)	606	1,6180	625	1,6662
WMS_getMap (tiger)	580	1,7667	586	1,6785
WMS_getMap_multilayer_3_svg (sf)	943	1,7620	1001	1,6722
WMS_getMap_multilayer_3_svg (tiger)	2412	1,7518	2610	1,6360
Totale	1134	6,9874	1205	6,5174

Distribuire l'applicazione tra più Availability Zone non introduce latenze

Partizionato VS Replicato - 2 workspace

La distribuzione mediante partizionamento è più efficiente nel caso di richieste riferite a workspace diverse?

- Simulati 16 utenti contemporanei

Richiesta	DB partizionato		DB replicato	
	Lat. media (ms)	Throughput (rich/sec)	Lat. media (ms)	Throughput (rich/sec)
WMS_getMap (sf)	524	3,4943	818	2,7626
WMS_getMap (tiger)	577	3,4895	807	2,7604
WMS_getMap_multilayer_3_svg (sf)	975	3,4847	1322	2,7477
WMS_getMap_multilayer_3_svg (tiger)	2516	3,4379	2870	2,7571
Totale	1146	13,7263	1452	10,8429

Con chiamate relative a dati in differenti workspace la distribuzione mediante partizionamento risulta più efficiente

Partizionato VS Replicato - 1 workspace

La distribuzione mediante replicazione è più efficiente nel caso di richieste riferite sempre alla stessa workspace?

- Simulati 16 utenti contemporanei

Richiesta	DB partizionato		DB replicato	
	Lat. media (ms)	Throughput (rich/sec)	Lat. media (ms)	Throughput (rich/sec)
WMS_getMap (tiger)	634	3,2172	1021	2,6029
WMS_getMap_multilayer_3_svg (tiger)	2972	3,1615	3227	2,5524
WMS_getMap_multilayer_cql (tiger)	684	3,2058	925	2,5952
WMS_getMap_multilayer_filter (tiger)	693	3,1974	989	2,5944
Totale	1248	12,5958	1542	10,1734

Distribuzione mediante partizionamento nuovamente più efficiente

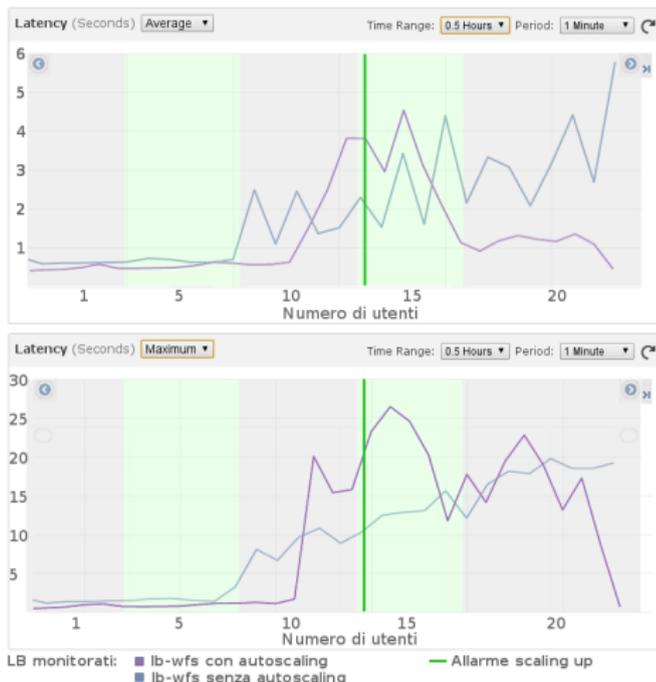
Per piccole distribuzioni il middleware non introduce benefici

Test sullo *scaling out*

Lo *scaling out* riduce la latenza quando essa diventa eccessiva?

Soglie *scaling*:

- *scaling out*: lat. > 15 s (MAX)



Latenza ridotta dallo *scaling out*

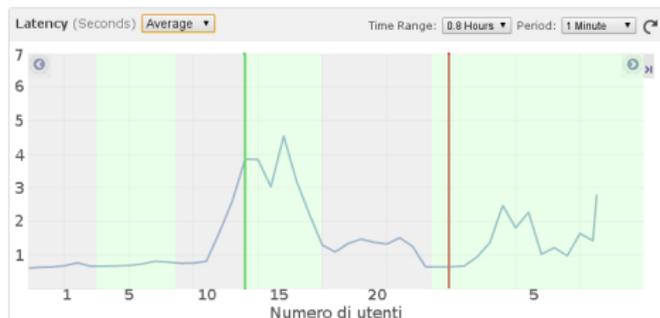
Ritardo tra allarme ed effettiva diminuzione della latenza

Test sullo *scaling in*

Vengono liberate le risorse quando non sono più necessarie?

Soglie *scaling*:

- *Scaling out*: lat. > 15 s (MAX)
- *Scaling in*: lat. < 1 s (MED)



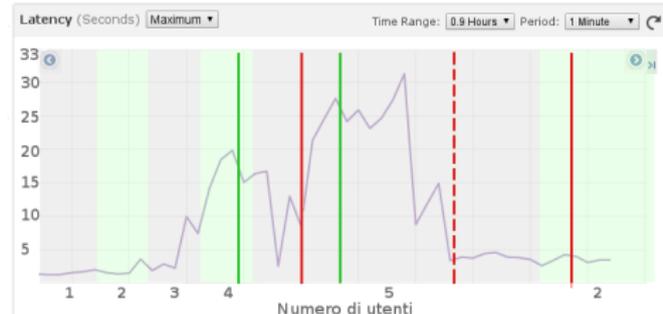
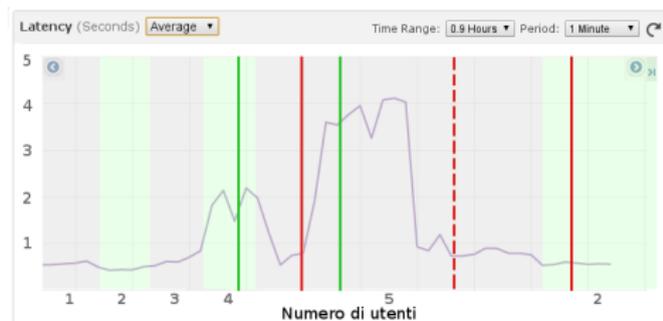
Tempo attivazione istanza > Tempo disattivazione istanza

Test sullo *scaling in* - oscillazione

Importante utilizzare soglie di *scaling* corrette!

Soglie *scaling*:

- *scaling out*: lat. > 10 s (MAX)
- *scaling in 1*: lat. < 1 s (MED)
- *scaling in 2*: lat. < 0.6 s (MED)



Load balancing monitorati: ■ lb-wms con autoscaling

LB monitorati: ■ lb-wms con autoscaling

— Allarme scaling up

— Allarme scaling down

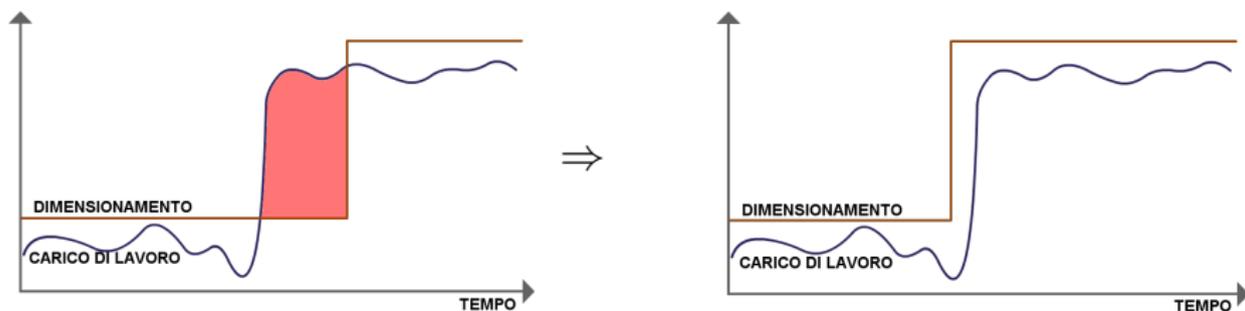
Oscillazione dovuta ad una soglia di *scaling in* troppo alta

Conclusioni

- La latenza si rivela una buona metrica per determinare se il dimensionamento è coerente con il carico di lavoro
- Rendere l'architettura più affidabile mediante una sua distribuzione su più Availability Zone non ne pregiudica le prestazioni
- Nel caso di richieste fatte a più workspace, la distribuzione mediante partizionamento si rileva più efficiente di quella realizzata mediante replicazione
- Lo *scaling out* è una operazione molto più onerosa, in termini di tempo e risorse, dello *scaling in*
- Difficile determinare soglie di *scaling* corrette

Scomposizione servizi

- Predire i picchi di carico in modo da anticipare lo scaling dei servizi



- Testare la distribuzione mediante replicazione con *scaling out* di dimensioni maggiori