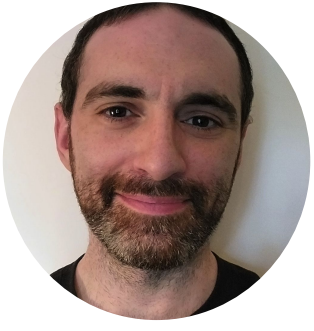
Interlogica

CODING THE FUTURE

Akai: Serverless Property Management System

Mi presento



Carlo Martini

DevOps Engineer
& Cloud Solution Specialist

In Interlogica mi occupo di:

- Ottimizzare processi e sistemi adottando un approccio DevOps.
- Sviluppare soluzioni cloud con tecnologie IaC ed con un focus su architetture serverless e sicurezza IT

Interlogica

Coding the Future

Un ecosistema agile proiettato nel futuro

Siamo una Tech Company specializzata nell'accompagnare i partner in percorsi coerenti di trasformazione digitale.

Dal 1995 disegniamo soluzioni software studiate intorno ai bisogni delle persone, generando valore condiviso e massimizzando la resa sul mercato delle attività delle realtà con cui collaboriamo.

Interlogica

Akai
SMART PMS

- **Soluzione integrata (Property Management System + Booking Engine):** preventivi ed offerte, prenotazioni, alloggi, reportistica, dati cliente, servizi, check-in / check-out, pagamenti, revenue management

- **Open Air (camping / glamping)**: scala (min. 10.000 utenti contemporanei) + variazioni (ogni alloggio / piazzola fa storia a sé)

- **Interfaccia verso…** ISTAT, Polizia di Stato, Channel Manager (Phobs), Data Intelligence (H-Benchmark), ERP (Zucchetti), stampanti fiscali, piattaforme prenotazione (PinCamp)

Interlogica

Principi Infrastrutturali

- **Multi-tenancy:** risorse (rete, computazione e storage) separate per ogni cliente
  (white paper *SaaS Tenant Isolation Strategies*)

- **Continuous Integration/Deploy**: patches, hotfixes, sviluppo

- **Event-Driven Architecture (“Serverless”)**: provisioning di risorse secondo necessità -> Reliability (Continuous Operations) -> Service Level Agreement

- **Continuous Monitoring**: *if things go south…*

- **Secure By Design**: layer di sicurezza per ogni componente
  (white paper *Building Security from the Ground up with Secure by Design*)

Booking Engine
(front-end auto-scalabile)

- Applicazione **Angular**
- Distribuzioni **Cloudfront** e **Bucket S3** come origin
- Reliability: disponibilità globale (caching su **edge locations**)
- Sicurezza: **Web Application Firewall**; **Origin Access Control** (Authorization header, protocollo SigV4); **Server-side encryption** (AES-256)
- Monitoraggio: allarmi Cloudwatch (**HealthCheckStatus** e **TotalErrorRate)** con notifica su topic **Simple Notification Service (SNS)**

Componenti applicativi
(back-end auto-scalabile)

- Gestionale PMS (PHP / Symfony)
- oAuth2 (PHP / Symfony)
- Revenue Manager (Python)
- Real-Time APIs (Mercure)
- ProxySQL (connection pooling/multiplexing;
  splitting lettura / scrittura delle query su DB cluster)

Interlogica

Componenti applicativi
(back-end auto-scalabile)

- **Elastic Container Service (Fargate compute engine)**:
  cluster (= tenant), servizi (= componenti), task (= gruppo logico di container)

- **Auto-scaling (orizzontale)**: gestito dal bilanciatore di carico (**Application Load Balancer**) in base ad una **policy (“Target Tracking”) di auto-scaling:** consumo medio CPU (%) e consumo medio memoria (%)

- **Scaling (verticale)**: in occasioni specifiche; da tarare sull’applicativo.

- **Approfondimento consigliato**:
  https://nathanpeck.com/amazon-ecs-scaling-best-practices/

Componenti applicativi
(back-end auto-scalabile)

- **Deployment** mediante **rolling update** e **circuit breaker**

- **Intra-comunicazione tra container:**
  - stesso task: port mapping su localhost (**networking awsvpc**, stessa ENI)
  - task (servizi) diversi: rete virtuale con namespace condiviso, proxy container sia nel task client che in quello server (**Service Connect**)

- **IP statico outbound (stampanti fiscali)**: NAT Gateway

- **Crittografia at rest** (AES-256) su storage effimero

Monitoring ECS

- **Monolog** (libreria Symfony per logging) configurata per scrivere su specifici **Cloudwatch log groups**

- **Cloudwatch Metric Filter** su pattern di stringe. Esempio:

  *[(w1=\"*CRITICAL*\") && (w1!=\"*Errore da Ignorare*\")]*

- **Allarme Cloudwatch** (azione = topic SNS) basata sul metric filter

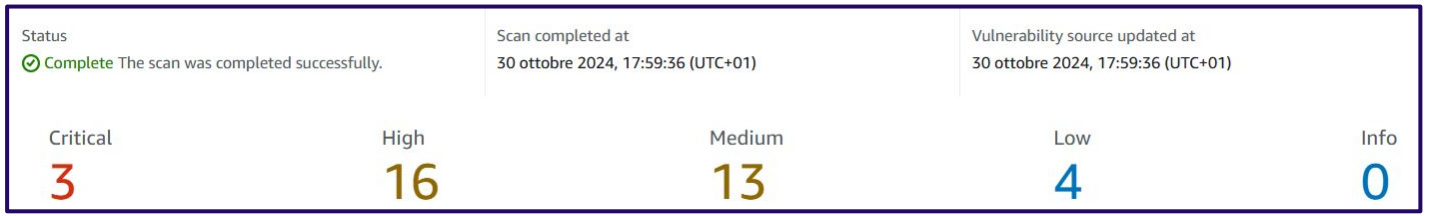
- Questione aperta: **monitoring della qualità del bilanciamento**. Possibile visualizzare in near-real-time le metriche di CPU e memoria dei singoli container (abilitando **Container Insights**), ma difficile trarne metriche per analisi temporale

Interlogica

Immagini Docker

- **Elastic Container Registry**

- **Scan on Push** →
  Rilevato con regola **Event Bridge** (event pattern su livello di vulnerabilità) →
  Allarme **Cloudwatch** →
  Notifica su **topic SNS**



Interlogica

Storage auto-scalabile

- **Elastic File System:** Network File System (versione 4)
- **Disponibilità regionale** (AZ multiple) e **replica automatica** (solo lettura) su altra regione
- Sicurezza: **crittografia at rest** (AES-256)
- **Modalità di Throughput:**
  default in Terraform = Bursting (storage ∝ throughput)
  default in AWS = Elastic (IOPS ∝ frequenza di accesso)

Database auto-scalabile

- **RDS Aurora MySQL Cluster**

- **Provisioned**: istanza di lettura-scrittura (primaria) e **repliche di sola lettura** (auto-scalabilità orizzontale). **Policy (“Target Tracking”) di auto-scaling:** consumo medio CPU (%) e media di connessioni (#)

- **Serverless v2**: auto-scalabilità verticale scrittura/lettura. Non ancora adottato per Akai. Motivo: variabilità di carico ridotta. Vedi:

  https://aws.amazon.com/blogs/database/evaluating-the-right-fit-for-your-amazon-aurora-workloads-provisioned-or-serverless-v2/

  https://www.keisuke.dev/2024/03/13/study_aws_aurora_serverless_v2.html

Database auto-scalabile

- Attenzione a…

- Il parametro di **max_connections** del MySQL di default è dinamico (dipendente dal tipo di istanza) e in teoria non va cambiato (*If you need more connections, we encourage you to upgrade to a larger RDS instance size*). Ma… Osservato un **sotto-utilizzo di risorse fisiche (a limite raggiunto)**, che per altro impediva l'auto-scalabilità basata sul consumo di CPU

- **Stessa classe di istanza e dimensionamento per tutte le componenti del cluster:**
  - Failover automatico: promossa ad istanza primaria una replica (sizing inadeguato?)
  - Scale-up dell'istanza primaria implica che… non sarà più primaria! Viene fatto failover (promozione a writer) di una delle repliche di lettura.

- **TTL (5 secondi) dei record DNS:** usati per failover, scaling di istanza, bilanciamento, quindi attenzione (= azzerare?) altre eventuali cache DNS

Database query splitting

- Come ottenere il **read/write splitting**?
- Ipotesi scartate: estensione PECL (porting per PHP 7/8 di mysqlnd_ms), ORM di Symfony (Doctrine), RDS local read replica write forwarding (non supporta “serializable” come livello di isolamento delle transazioni)
- Soluzione: **servizio ECS con ProxySQL**. Bonus: **pooling connessioni**
- Questione aperta: **bilanciamento del reader endpoint** non ottimale tra le repliche (anche con cache DNS disabilitata; vedi slide precedente)

Status	Role	Engine	Region & AZ	Size	CPU	Current activity
Available	Regional cluster	Aurora MySQL	eu-west-1	3 instances	-	-
Available	Writer instance	Aurora MySQL	eu-west-1b	db.r5.xlarge	5.73%	0.00 sessions
Available	Reader instance	Aurora MySQL	eu-west-1c	db.r5.xlarge	4.99%	0.14 sessions
Available	Reader instance	Aurora MySQL	eu-west-1a	db.r5.xlarge	15.57%	0.44 sessions

Database schema migrations

- Come modificare lo schema del database **senza downtime**?

- **Principio**: **prima la migration; poi il rilascio del nuovo codice**. MA… Quando devo cancellare una tabella o colonna servono due deployment (il primo non si aspetta più la presenza di quell'elemento; il secondo lo cancella attivamente)

- Tecnicamente possibile "SSH" (mediante **AWS Systems Manager**) per eseguire comandi nei container (nel nostro caso Symfony CLI per Doctrine ORM), ma preferibile automazione con uso di **one-off tasks**

https://engineering.resolvergroup.com/2020/09/running-database-migrations-on-deployment-for-fargate-containers/

Sessioni utente

- **Cluster di nodi Elasticache** (Memcached) per ogni tenant
- Serverless disponibile da fine 2023 → gestire **crittografia in transito**
- **Estensione PHP** (Cluster Client) per auto-discovery dei nodi e configurazione di **Symfony service**: https://symfony.com/doc/current/session.html#store-sessions-in-a-key-value-database-redis

Eventi programmati

- Passaggio da **Rundeck Runbook Automation** (Pagerduty) su EC2 ad **EventBridge Scheduler** (AWS) serverless

- Invocazioni di **lambda functions** (Go, Python/Boto3, Node) con payload diversificati

- **Comandi Symfony** su task ECS, leggendo l'output e stampandolo (in modo unificato) su log Cloudwatch: utilizzo di una **connessione WebSocket** (ispirati da questo thread: https://github.com/boto/boto3/issues/3496)

```
connection = websocket.create_connection(session['streamUrl'])

AgentMessage = c.Struct(
    "HeaderLength" / c.Int32ub,
    "MessageType" / c.PaddedString(32, "ascii"),
    "SchemaVersion" / c.Int32ub,
    "CreateDate" / c.Int64ub,
    "SequenceNumber" / c.Int64sb,
    "Flags" / c.Int64ub,
    "MessageID" / c.Array(16, c.Byte),
    "PayloadDigest" / c.Array(32, c.Byte),
    "PayloadType" / c.Int32ub,
    "PayloadLength" / c.Int32ub,
    # il messaggio può contenere caratteri accentati
    "Payload" / c.PaddedString(c.this.PayloadLength, 'utf8')
)
```


Mailing system

- Identità **SES (Simple Email System)**
- Autenticazione **DKIM**
- **Configuration Set** per tracking di eventi (hard bounce, reject…) su topic SNS.
  Subscription su endpoint HTTPS del PMS

Security automatica

- **Amazon Inspector** (ECR, EC2, Lambda) con notifiche (mediante EventBridge)
- **TFsec / Trivy** (con **AWS Checks**)

Interlogica

Eccezioni all'uso di AWS

Lessons *learned!*

- **Terraform** anziché **CloudFormation**
- **GitLab** anzichè **CodeCommit / CodePipeline**
  (pipeline basate su **AWS CLI** per deploy Cloudfront, Lambdas, ECR/ECS)
- **Loader.io** anziché **Distributed Load Testing (AWS Solutions Library)**
- **TFsec / Trivy** (con **AWS Checks**) anziché **Trusted Advisor**
- **Razionale**: consistency aziendale > consistency di progetto

Interlogica

Prospettive future

- Serverless Gitlab Runners (ECS/Fargate)
- **Serverless Elasticache**
- **Proxy RDS** (= multi-plexing / pooling connessioni nativo di AWS) e gestione read/write splitting da applicativo

Grazie!

INTERLOGICA SRL SOCIETÀ BENEFIT

VIA MIRANESE, 91/4 - 30174 VENEZIA-MESTRE (VE)
TEL: +39 041 5354800 - E-MAIL: info@interlogica.it
www.interlogica.it

ISO 9001:2015

ISO/IEC 27001