**Escalabilidad, DevOps Avanzado y Roadmap - TurisGal**

**1. ESCALABILIDAD Y ARQUITECTURA DISTRIBUIDA**

**Microservicios Avanzados con Kubernetes**

# k8s/microservices-architecture.yaml

apiVersion: v1

kind: Namespace

metadata:

name: turisgal-production

---

# User Service

apiVersion: apps/v1

kind: Deployment

metadata:

name: user-service

namespace: turisgal-production

spec:

replicas: 3

selector:

matchLabels:

app: user-service

template:

metadata:

labels:

app: user-service

version: v1

spec:

containers:

- name: user-service

image: turisgal/user-service:latest

ports:

- containerPort: 3001

env:

- name: DATABASE\_URL

valueFrom:

secretKeyRef:

name: database-secret

key: user-db-url

- name: REDIS\_URL

valueFrom:

secretKeyRef:

name: redis-secret

key: url

resources:

requests:

memory: "256Mi"

cpu: "250m"

limits:

memory: "512Mi"

cpu: "500m"

livenessProbe:

httpGet:

path: /health

port: 3001

initialDelaySeconds: 30

periodSeconds: 10

readinessProbe:

httpGet:

path: /ready

port: 3001

initialDelaySeconds: 5

periodSeconds: 5

---

# Checkin Service

apiVersion: apps/v1

kind: Deployment

metadata:

name: checkin-service

namespace: turisgal-production

spec:

replicas: 5 # Más instancias para servicio crítico

selector:

matchLabels:

app: checkin-service

template:

metadata:

labels:

app: checkin-service

version: v1

spec:

containers:

- name: checkin-service

image: turisgal/checkin-service:latest

ports:

- containerPort: 3002

env:

- name: OCR\_SERVICE\_URL

value: "http://ocr-service:3010"

- name: FACE\_MATCHING\_URL

value: "http://face-matching-service:3011"

resources:

requests:

memory: "512Mi"

cpu: "500m"

limits:

memory: "1Gi"

cpu: "1000m"

---

# Horizontal Pod Autoscaler

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: checkin-service-hpa

namespace: turisgal-production

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: checkin-service

minReplicas: 3

maxReplicas: 20

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 70

- type: Resource

resource:

name: memory

target:

type: Utilization

averageUtilization: 80

behavior:

scaleUp:

stabilizationWindowSeconds: 60

policies:

- type: Percent

value: 100

periodSeconds: 15

scaleDown:

stabilizationWindowSeconds: 300

policies:

- type: Percent

value: 10

periodSeconds: 60

---

# Service Mesh con Istio

apiVersion: networking.istio.io/v1alpha3

kind: VirtualService

metadata:

name: turisgal-routing

namespace: turisgal-production

spec:

http:

- match:

- uri:

prefix: /api/users

route:

- destination:

host: user-service

port:

number: 3001

weight: 90

- destination:

host: user-service-canary

port:

number: 3001

weight: 10

- match:

- uri:

prefix: /api/checkin

route:

- destination:

host: checkin-service

port:

number: 3002

fault:

delay:

percentage:

value: 0.1

fixedDelay: 5s

retries:

attempts: 3

perTryTimeout: 2s

**Event-Driven Architecture**

// src/events/EventBus.ts

export class EventBus {

private subscribers: Map<string, EventHandler[]> = new Map();

private eventStore: EventStore;

private messageQueue: MessageQueue;

constructor() {

this.eventStore = new PostgreSQLEventStore();

this.messageQueue = new RedisMessageQueue();

this.setupDeadLetterQueue();

}

// Publish events con garantía de entrega

async publish(event: DomainEvent): Promise<void> {

const eventId = crypto.randomUUID();

const enrichedEvent = {

...event,

id: eventId,

timestamp: new Date(),

version: 1,

correlationId: event.correlationId || eventId

};

try {

// 1. Guardar en event store (outbox pattern)

await this.eventStore.append(enrichedEvent);

// 2. Publicar a message queue

await this.messageQueue.publish(event.type, enrichedEvent);

// 3. Notificar subscribers locales

await this.notifyLocalSubscribers(enrichedEvent);

// 4. Marcar como publicado en event store

await this.eventStore.markAsPublished(eventId);

} catch (error) {

await this.handlePublishError(enrichedEvent, error);

throw error;

}

}

// Subscribe con diferentes strategies

subscribe(eventType: string, handler: EventHandler, options?: SubscriptionOptions): void {

if (!this.subscribers.has(eventType)) {

this.subscribers.set(eventType, []);

}

const wrappedHandler = this.wrapHandler(handler, options);

this.subscribers.get(eventType)!.push(wrappedHandler);

// Setup message queue subscription

this.messageQueue.subscribe(eventType, wrappedHandler, options);

}

private wrapHandler(handler: EventHandler, options?: SubscriptionOptions): EventHandler {

return async (event: DomainEvent) => {

const handlerName = handler.constructor.name;

const startTime = Date.now();

try {

// Idempotency check

if (options?.idempotent) {

const alreadyProcessed = await this.checkIfProcessed(event.id, handlerName);

if (alreadyProcessed) {

console.log(`Event ${event.id} already processed by ${handlerName}`);

return;

}

}

// Circuit breaker

if (options?.circuitBreaker) {

await this.checkCircuitBreaker(handlerName);

}

// Execute handler

await handler(event);

// Mark as processed

if (options?.idempotent) {

await this.markAsProcessed(event.id, handlerName);

}

// Metrics

this.recordHandlerSuccess(handlerName, Date.now() - startTime);

} catch (error) {

this.recordHandlerError(handlerName, error);

// Retry logic

if (options?.retry) {

await this.scheduleRetry(event, handler, options.retry);

} else {

await this.sendToDeadLetterQueue(event, error);

}

throw error;

}

};

}

}

// Domain Events

export class CheckInCompletedEvent implements DomainEvent {

readonly type = 'checkin.completed';

constructor(

public readonly bookingId: string,

public readonly userId: string,

public readonly propertyId: string,

public readonly checkInTimestamp: Date,

public readonly verificationStatus: string,

public readonly correlationId?: string

) {}

}

export class PaymentProcessedEvent implements DomainEvent {

readonly type = 'payment.processed';

constructor(

public readonly paymentId: string,

public readonly bookingId: string,

public readonly amount: number,

public readonly currency: string,

public readonly status: 'success' | 'failed',

public readonly correlationId?: string

) {}

}

// Event Handlers

export class CheckInEventHandlers {

constructor(

private emailService: EmailService,

private notificationService: NotificationService,

private analyticsService: AnalyticsService,

private inventoryService: InventoryService

) {}

@EventHandler('checkin.completed')

async onCheckInCompleted(event: CheckInCompletedEvent): Promise<void> {

// Enviar email de bienvenida

await this.emailService.sendWelcomeEmail(event.userId, event.bookingId);

// Actualizar inventario

await this.inventoryService.markRoomAsOccupied(event.propertyId, event.bookingId);

// Registrar métricas

await this.analyticsService.recordCheckIn(event);

// Programar recordatorio de check-out

await this.notificationService.scheduleCheckOutReminder(event.bookingId);

}

@EventHandler('checkin.verification\_failed')

async onVerificationFailed(event: VerificationFailedEvent): Promise<void> {

// Notificar al property owner

await this.notificationService.notifyPropertyOwner(

event.propertyId,

'verification\_failed',

{ bookingId: event.bookingId, reason: event.reason }

);

// Escalate to manual review

await this.manualReviewService.createReviewTask(event);

}

}

**Caching Distribuido y CDN**

// src/cache/DistributedCacheService.ts

export class DistributedCacheService {

private localCache: NodeCache;

private redisCluster: Redis.Cluster;

private cdnService: CDNService;

constructor() {

this.localCache = new NodeCache({ stdTTL: 300 });

this.redisCluster = new Redis.Cluster([

{ host: 'redis-node-1', port: 6379 },

{ host: 'redis-node-2', port: 6379 },

{ host: 'redis-node-3', port: 6379 }

]);

this.cdnService = new CloudFlareCDNService();

}

// Cache strategy por tipo de datos

async get<T>(key: string, fetchFn?: () => Promise<T>, options?: CacheOptions): Promise<T | null> {

const cacheStrategy = this.determineCacheStrategy(key, options);

switch (cacheStrategy) {

case 'local\_only':

return await this.getFromLocal(key, fetchFn, options);

case 'distributed\_only':

return await this.getFromDistributed(key, fetchFn, options);

case 'multi\_tier':

return await this.getMultiTier(key, fetchFn, options);

case 'cdn\_static':

return await this.getFromCDN(key, fetchFn, options);

default:

return await this.getMultiTier(key, fetchFn, options);

}

}

private async getMultiTier<T>(key: string, fetchFn?: () => Promise<T>, options?: CacheOptions): Promise<T | null> {

// Nivel 1: Cache local (más rápido)

let value = this.localCache.get<T>(key);

if (value !== undefined) {

return value;

}

// Nivel 2: Redis distribuido

const redisValue = await this.redisCluster.get(key);

if (redisValue) {

try {

value = JSON.parse(redisValue);

// Backfill cache local

this.localCache.set(key, value, Math.min(options?.ttl || 3600, 300));

return value;

} catch (error) {

console.error('Error parsing Redis value:', error);

}

}

// Nivel 3: CDN para assets estáticos

if (options?.cdn && key.includes('static/')) {

const cdnValue = await this.cdnService.get(key);

if (cdnValue) {

// Cache en Redis y local

await this.set(key, cdnValue, options);

return cdnValue;

}

}

// Nivel 4: Fetch from source

if (fetchFn) {

value = await fetchFn();

if (value !== null && value !== undefined) {

await this.set(key, value, options);

}

return value;

}

return null;

}

// Cache warming inteligente

async warmCache(warmingPlan: CacheWarmingPlan): Promise<void> {

const warmingTasks = warmingPlan.tasks.map(async (task) => {

try {

switch (task.type) {

case 'popular\_properties':

await this.warmPopularProperties();

break;

case 'user\_preferences':

await this.warmUserPreferences();

break;

case 'static\_content':

await this.warmStaticContent();

break;

case 'geo\_data':

await this.warmGeoData();

break;

}

} catch (error) {

console.error(`Cache warming failed for ${task.type}:`, error);

}

});

await Promise.allSettled(warmingTasks);

}

// Cache invalidation patterns

async invalidatePattern(pattern: string, options?: InvalidationOptions): Promise<vo