UNIVERSIDAD POPULAR DEL CESAR
ESPECIALIZACION DE INGENIEIRA DE SOFTWARE
GUÍA DE ACTIVIDADES PARA LA CONSTRUCCIÓN DE CONOCIMIENTO

NOMBRE DE LA ASIGNATURA	Patrones De Diseño de Software							
NOMBRE DE LA ACTIVIDAD	Introducción a UML y Diseño Orientado a Objetos							
TIPO DE ACTIVIDAD	Sincrónica		Asincrónica	x	Individual	x	Grupal	

TEMÁTICA REQUERIDA PARA LA ACTIVIDAD	**OBJETIVOS**
Unidad 3. Patrones de diseño creacionales. Unidad 2. Principios de diseño de software Unidad 1. Principios de POO y UML	Evaluar la capacidad para diseñar un API aplicando el patrón de diseño Factory Method, con el propósito de crear una solución escalable, flexible y mantenible.
COMPETENCIAS	**INSUMOS PARA EL DESARROLLO DE LA ACTIVIDAD / REFERENCIAS BIBLIOGRÁFICAS**
• Aplicación de principios de diseño SOLID	• Presentaciones unidad 1, 2 y 3 • Código de ejemplo desarrollado en aula • Bibliografía recomendada

CONOCIMIENTOS PREVIOS REQUERIDOS

Conceptos fundamentales de POO, UML, Principios SOLID, Factory Method Design Pattern

ESPECIFICACIONES DE LA ACTIVIDAD

Caso Práctico: API de Aprovisionamiento de Máquinas Virtuales Multi-Cloud

Diseñar e implementar una API REST de aprovisionamiento de infraestructura cloud que permita crear máquinas virtuales (VM) junto con sus recursos asociados de red y almacenamiento, garantizando coherencia entre los recursos de un mismo proveedor y validación de dependencias cruzadas.

El sistema debe admitir múltiples proveedores de nube (AWS, Azure, GCP y On-Premise) y permitir la creación de tres tipos de máquinas virtuales mediante el uso del patrón **Builder** controlado por un **Director**:

1. Standard VM
2. VM Optimizada en Disco
3. VM Optimizada en Memoria

En esta ocasión, se sebe tener en cuenta que se han agregado nuevos parámetros de configuración a los diferentes recursos VM, Network y Storage en cada proveedor, algunos son de carácter obligatorio, mientras que otros son opcionales.

Parámetros adicionales de configuración por recurso y proveedor

- **VirtualMachine**

Atributo	Descripción	Obligatorio	Ejemplo
provider	Proveedor de nube	☑	"AWS"
vcpus	Núcleos virtuales asignados	☑	2
memoryGB	Memoria RAM asignada	☑	4
memoryOptimization	Optimización de memoria	✗	true
diskOptimization	Optimización de disco	✗	false
keyPairName	Clave SSH o autenticación	✗	"default-key"

- **Network**

Atributo	Descripción	Obligatorio	Ejemplo
region	Región de red	☑	"us-east-1"
firewallRules	Reglas de seguridad	✗	["HTTP", "SSH"]
publicIP	IP pública asignada	✗	true

- **Storage**

Atributo	Descripción	Obligatorio	Ejemplo
region	Región del almacenamiento	☑	"us-east-1"
iops	Rendimiento del disco	✗	3000

Tipos de Máquina (Director)

El **Director** define la política de construcción y los valores de recursos de cómputo (vCPU y memoria RAM):

Amazon AWS:

1. **General Purpose (Standard)**
   - t3.medium: 2 vCPU, 4 GiB RAM
   - m5.large: 2 vCPU, 8 GiB RAM
   - m5.xlarge: 4 vCPU, 16 GiB RAM
2. **Memory-Optimized**
   - r5.large: 2 vCPU, 16 GiB RAM
   - r5.xlarge: 4 vCPU, 32 GiB RAM
   - r5.2xlarge: 8 vCPU, 64 GiB RAM
3. **Compute-Optimized**
   - c5.large: 2 vCPU, 4 GiB RAM
   - c5.xlarge: 4 vCPU, 8 GiB RAM
   - c5.2xlarge: 8 vCPU, 16 GiB RAM

Microsoft Azure

Azure no siempre publica en forma “nombre + vCPU/RAM fijos” en el mismo sitio, pero puedes usar familias conocidas:

1. **Standard / General Purpose**

    - D2s_v3: 2 vCPU, 8 GiB RAM
    - D4s_v3: 4 vCPU, 16 GiB RAM
    - D8s_v3: 8 vCPU, 32 GiB RAM

2. **Memory-Optimized**

    - E2s_v3: 2 vCPU, 16 GiB RAM
    - E4s_v3: 4 vCPU, 32 GiB RAM
    - E8s_v3: 8 vCPU, 64 GiB RAM

3. **Compute-Optimized**

    - F2s_v2: 2 vCPU, 4 GiB RAM
    - F4s_v2: 4 vCPU, 8 GiB RAM
    - F8s_v2: 8 vCPU, 16 GiB RAM

Google Cloud Platform (GCP):

1. **Standard / General Purpose (E2 standard / N2 standard)**

    - e2-standard-2: 2 vCPU, 8 GiB RAM
    - e2-standard-4: 4 vCPU, 16 GiB RAM
    - e2-standard-8: 8 vCPU, 32 GiB RAM

2. **Memory-Optimized (High memory / N-family highmem)**

    - n2-highmem-2: 2 vCPU, 16 GiB RAM
    - n2-highmem-4: 4 vCPU, 32 GiB RAM
    - n2-highmem-8: 8 vCPU, 64 GiB RAM

3. **Compute-Optimized (High CPU)**

    - n2-highcpu-2: 2 vCPU, 2 GiB RAM
    - n2-highcpu-4: 4 vCPU, 4 GiB RAM
    - n2-highcpu-8: 8 vCPU, 8 GiB RAM

On-Premise (VM Flavors / Simulación):

Para on-premise (por ejemplo, máquinas virtuales en un hipervisor KVM, VMWare), podrías definir "flavors" similares:

1. **Standard**

   - onprem-std1: 2 vCPU, 4 GiB RAM
   - onprem-std2: 4 vCPU, 8 GiB RAM
   - onprem-std3: 8 vCPU, 16 GiB RAM

2. **Memory-Optimized**

   - onprem-mem1: 2 vCPU, 16 GiB RAM
   - onprem-mem2: 4 vCPU, 32 GiB RAM
   - onprem-mem3: 8 vCPU, 64 GiB RAM

3. **Compute-Optimized**

   - onprem-cpu1: 2 vCPU, 2 GiB RAM
   - onprem-cpu2: 4 vCPU, 4 GiB RAM
   - onprem-cpu3: 8 vCPU, 8 GiB RAM

Requerimientos funcionales (RF)

RF1. La API debe permitir crear máquinas virtuales en diferentes proveedores (AWS, Azure, GCP, On-Premise).
RF2. Cada VM debe asociarse a una red y almacenamiento del mismo proveedor.
RF3. Debe existir un **Director** que orqueste el proceso de construcción (Builder) según el tipo de VM.
RF4. El Director debe asignar **valores de vCPU y memoria RAM** según el tipo de máquina y proveedor.
RF5. Los recursos deben validarse para garantizar coherencia de región y proveedor.

Requerimientos no funcionales (RNF)

RNF1. Modularidad: separación clara entre la lógica de creación (Factory), ensamblado (Builder) y control (Director).
RNF2. Extensibilidad: la incorporación de nuevos tipos de VM o proveedores no debe requerir cambios en el código existente.
RNF3. Validación cruzada: coherencia de proveedor y región entre los recursos creados.
RNF4. Escalabilidad: la arquitectura debe soportar múltiples despliegues simultáneos.
RNF5. Legibilidad: el código debe ser entendible como material de enseñanza de patrones combinados.

Valor didáctico de este incremento

- **Builder**: permite parametrizar la construcción paso a paso de recursos con configuraciones diferentes.
- **Director**: controla la lógica de negocio (qué valores asignar a CPU y RAM según el tipo de VM).
- **Abstract Factory**: garantiza coherencia entre los recursos del mismo proveedor.
- **Factory Method**: se mantiene en cada fábrica concreta para crear los objetos de bajo nivel.

• **Validación cruzada y coherencia**: refuerza principios de **encapsulación de variaciones** y **programación a interfaces**.	
RECOMENDACIONES / OBSERVACIONES	Para el diseño UML del Diagrama de clases se sugiere utilizar cualquiera de las siguientes herramientas: StartUML, PlantUML, Draw.io, GenMyModel o Visual Paradigm. Puede utilizar el lenguaje de programación de su preferencia, preferiblemente Java, es opcional el desarrollo de interfaces Graficas de Usuario.

Elaboro: Ing. Jairo Seoanes, Msc Ingeniería de Sistemas y Computación