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	Patrones de diseño creacionales							
TIPO DE ACTIVIDAD	Sincrónica		Asincrónica	x	Individual	x	Grupal	

TEMÁTICA REQUERIDA PARA LA ACTIVIDAD	**OBJETIVOS**
Unidad 2. Patrones creacionales Unidad 1. Fundamentos avanzados de diseño de software	Evaluar la capacidad para diseñar un API aplicando patrones de diseño creacionales, 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 • Código de ejemplo desarrollado en aula • Bibliografía recomendada

CONOCIMIENTOS PREVIOS REQUERIDOS

Conceptos fundamentales de POO, UML, Principios SOLID, Patrones de diseño creacionales

ESPECIFICACIONES DE LA ACTIVIDAD

Caso Práctico: API de Aprovisionamiento de recursos de nube (cloud) - Multicloud

Una empresa de infraestructura distribuida necesita un API unificado que le permita aprovisionar recursos de nube desde distintos tipos de proveedores (AWS | Azure | Google Cloud).

Los recursos de nube que se quieren aprovisionar son: discos de bloque (Storage), redes virtuales (SDN/Networking) e instancias de máquinas virtuales (VM).

Para cada uno de estos servicios cada proveedor maneja parámetros diferentes para su configuración. Tal como se muestra en las siguientes tablas:

Discos:

Atributo	Descripción	Obligatorio	Valores
volumenType	Si es un disco de uso estándar o si es optimizado	✅	**Aws**: gp2, io1 **Azure:** Standard_LRS, Premium_LRS **gcp**: pd-standard, pd-ssd
sizeGB	Capacidad máxima de almacenamiento	✅	50Gb
encrypted	Si aplica encriptamiento avanzado	❌	true / false
managedDisk	Si es autoadministrado	❌	true / false

autoDelete	Si se elimina automática al terminar las instancias	❌	true / false
region	Región del almacenamiento	✅	**Aws**: us-east-1, us-west-2 **Azure:** westcentralus, southcentralus **gcp**: us-central1, us-east1
tag	Nombre para identificar el recurso	✅	ssd-backup-dev
provider	Proveedor de nube	✅	"AWS"

Redes (Networking):

Atributo	Descripción	Obligatorio	Valores
region	Región de red	✅	**Aws**: us-east-1, us-west-2 **Azure:** westcentralus, southcentralus **gcp**: us-central1, us-east1
firewallRules	Reglas de seguridad	❌	["TCP:80", "TCP:22"," TCP:3306"]
CIDR	Bloque de direcciones ip disponibles	✅	130.0.0.0/16
tag	Nombre para identificar el recurso	✅	default-network
provider	Proveedor de nube	✅	"AWS"

Instancias (VM):

Atributo	Descripción	Obligatorio	Ejemplo
provider	Proveedor de nube	✅	"AWS"
vcpus	Núcleos virtuales asignados	✅	2
memoryOptimization	Optimización de memoria	❌	true/false
diskOptimization	Optimización de disco	❌	true/false
deleteProtection	Protección de terminación	❌	true/false
keyPairName	Clave SSH o autenticación	❌	"default-key"
tag	Nombre para identificar el recurso	✅	My-web-server
instanceType	Tipo de instancia	✅	
AMI	Imagen de maquina (SO)	✅	
publicIp	Si su dirección ip es publica o no	❌	tru/false
Storage/Disco	Disco asociado a la instancia	✅	Según el proveedor
Red/Network	Red en la cual se lanza la instancia	✅	Según el proveedor

La empresa requiere una solución extensible para integrar **nuevos proveedores en el futuro**, sin modificar la lógica central del API.

1. Requerimientos Funcionales (RF):

- **RF1**: La API debe permitir aprovisionar recursos simples (Discos y Redes) y también recursos complejos como máquinas virtuales (contiene disco y red), debe definir diferentes endpoint para cada caso.

  Para cada proveedor, la API debe invocar la lógica correspondiente de creación de los recursos usando los parámetros esperados por ese proveedor (validar coherencia).

  Para el aprovisionamiento de máquinas virtuales se debe garantizar que los recursos de red y discos asociados sean del mismo proveedor y garantizar coherencia de región.

La API debe devolver como resultado el recurso aprovisionado con sus parámetros, un estado de aprovisionamiento (success o error con causa detallada) y un id de aprovisionamiento para cada recurso aprovisionado exitosamente.

- **RF2**: La Api debe ofrecer algunas plantillas prediseñadas para la configuración rápida de recursos, para facilitar la configuración de estos.

  Se debe permitir aprovisionar recursos a partir de dichas plantillas (solamente modificando los valores deseados). Además, se puede alimentar el API con nuevas plantillas de configuración.

- **RF3**: La API debe mantener un log de las solicitudes de aprovisionamiento (sin exponer información sensible, solo con lo necesario).

- **RF4**: La API debe mantener un inventario de los recursos aprovisionados con información básica del recurso, adema fecha y hora de aprovisionamiento.

2. Requerimientos No Funcionales (RNF)

- **RNF1 – Extensibilidad:** La solución debe permitir incorporar nuevos proveedores sin modificar código existente.

- **RNF2 – Mantenibilidad:** Debe aplicarse **SOLID**, para no acoplar el controlador a implementaciones concretas y facilitar extensibilidad y mantenibilidad.

- **RNF3 – Seguridad:** Los parámetros sensibles (a su consideración) no deben aparecer en logs.

- **RNF4 – Compatibilidad:** Las solicitudes a la API deben ser en formato **JSON** independientemente del proveedor, así como sus respuestas.

- **RNF5** – Se deben incorporar patrones de diseño de software creacionales para promover el cumplimiento de principios SOLID y en consecuencia la reutilización, extensibilidad y mantenibilidad (flexibilidad).

- **RNF6 – Consistencia**: Se debe validar los datos de las solicitudes realizadas a la API para garantizar que los recursos se aprovisionen de manera consistente de acuerdo a los valores de los parámetros definidos por cada proveedor.

3. Entregables:

Documento pdf con los siguientes artefactos:

- Preliminares (presentación, etc)
- Diagrama de clases de la solución (modelo de dominio)
- Diagrama de arquitectura (componentes)
- Enlace del repositorio con acceso al código implementado (debe incluir readme con documentación de la solución)
- Sustentación de la solución (cada grupo tendrá 25 min par presentar los resultados de su trabajo)

RECOMENDACIONES / OBSERVACIONES	Para el diseño UML del Diagrama de clases se sugiere utilizar cualquiera de las siguientes herramientas: StartUML, PlantUML, Draw.io o Visual Paradigm. Puede utilizar el lenguaje de programación de su preferencia, preferiblemente Java, es opcional el desarrollo de interfaces Graficas de Usuario (front-end)

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