Universidad de la Ciudad de Aguascalientes

**Maestría en Ciencia de Datos** (RVOE 2727)

Materia : Ingeniería de datos

2. Diseño

2.1 Diseño de procesos propuestos.
2.1.1 Herramientas CASE para diseño.
**2.2 Diseño arquitectónico.**
2.3 Diseño de datos.
2.4 Diseño de interfaz de usuario.

**Definición:**

El diseño arquitectónico es el proceso de definir la estructura, componentes, módulos y relaciones del sistema.

En esta etapa, se toman decisiones importantes sobre cómo los diferentes elementos del sistema interactuarán entre sí para lograr los objetivos de manera eficiente y efectiva.

**Importancia:**

- **Claridad, eficiencia, eficacia y comprensión**: Proporciona una representación visual y estructural del sistema, lo que facilita la interpretación para los involucrados en el proyecto.
- **Colaboración**: Permite que múltiples equipos trabajen en un proyecto, dividiendo las actividades en módulos o componentes específicos.
- **Mantenimiento**: Un diseño bien planificado facilita las actualizaciones y correcciones en el ciclo de vida del proyecto.

**Elementos clave**:

- **Componentes**: Identificación de los elementos principales que compondrán el sistema.
- **Módulos**: División del sistema en unidades funcionales.
- **Conexiones**: Definición de cómo los componentes se comunicarán entre sí.
- **Patrones de diseño**: Utilización de estrategias arbitradas para resolver eventualidades comunes en el diseño.
- **Estándares y convenciones**: Uso de normas y buenas prácticas.

**Métodos y enfoques:**

- **Modelado arquitectónico**: Utilización de diagramas de arquitectura, como diagramas de componentes, diagramas de despliegue y diagramas de flujo de datos, para representar la estructura del sistema.

- **Metodologías arquitectónicas**: Ejemplos incluyen la arquitectura en capas, la arquitectura orientada a servicios (SOA) y la arquitectura de microservicios.

- **Principios de diseño**: Cumplir con principios como la separación de validaciones, consistencia y el acoplamiento.

**Metodología arquitectónica**: la arquitectura en capas.

La arquitectura en capas es un concepto fundamental en la ingeniería de procesos que se utiliza para diseñar sistemas de manera modular y escalable. Esta arquitectura se basa en la idea de dividir en capas o niveles, donde cada capa tiene una responsabilidad específica y se comunica con las capas adyacentes a través de interfaces bien definidas.

**Metodología arquitectónica**: la arquitectura en capas.

Principios fundamentales:

- **Separación de Responsabilidades**: Cada capa del sistema tiene una responsabilidad claramente definida y no debe interferir en las responsabilidades de otras capas. Esto facilita la modularidad y el mantenimiento del sistema.
- **Abstracción**: Cada capa oculta los detalles de implementación de las capas inferiores, lo que permite cambios en una capa sin afectar a las demás.
- **Interfaz Bien Definida**: Cada capa se comunica con las capas adyacentes a través de interfaces bien definidas y documentadas. Esto permite una comunicación clara y consistente entre las capas.

**Metodología arquitectónica**: la arquitectura en capas.

Ventajas:

- **Modularidad**: Facilita la creación de módulos independientes y reutilizables, lo que simplifica el desarrollo y el mantenimiento del software.
- **Escalabilidad**: Permite agregar o modificar capas según sea necesario para adaptarse a los cambios en los requisitos o el crecimiento del sistema.
- **Facilita la Colaboración**: Diferentes equipos pueden trabajar en capas diferentes de manera concurrente sin interferir en el trabajo de los demás.
- **Mantenimiento sencillo**: La corrección de errores y las actualizaciones se pueden realizar de manera aislada en una capa sin afectar al resto del sistema.

**Metodología arquitectónica**: la arquitectura en capas.

Observaciones:

- **Overhead de Comunicación**: La comunicación entre capas puede introducir cierto overhead en el rendimiento del sistema.
- **Diseño Inicial Complejo**: Definir las capas y las interfaces correctamente requiere un diseño inicial cuidadoso.
- **Rigidez Potencial**: Si no se planifica adecuadamente, la arquitectura en capas puede volverse rígida y dificultar la adaptación a cambios en los requisitos.

**Metodología arquitectónica**: la arquitectura en capas.

Ejemplos:

- **Aplicaciones Web**: En el desarrollo web, es común utilizar una arquitectura en capas que consta de una capa de presentación (frontend), una capa de lógica de negocio y una capa de acceso a datos (backend).
- **Sistemas de Gestión de Bases de Datos**: Los sistemas de gestión de bases de datos a menudo utilizan una arquitectura en capas que incluye una capa de presentación de consultas, una capa de motor de base de datos y una capa de almacenamiento físico.
- **Sistemas Empresariales**: Los sistemas empresariales suelen implementarse con una arquitectura en capas que separa la interfaz de usuario, la lógica de negocio y la capa de acceso a datos.
- **Sistemas Embebidos**: Incluso en sistemas embebidos, donde los recursos son limitados, se pueden aplicar principios de arquitectura en capas para mantener una separación clara de responsabilidades.

**Metodología arquitectónica**: la arquitectura en capas.

Nota:

*Es un enfoque fundamental en la ingeniería de software que se utiliza para diseñar sistemas de software modular y escalable. Ofrece ventajas como modularidad y escalabilidad, pero también presenta desafíos en términos de diseño y overhead de comunicación. Su aplicación varía según el tipo de sistema, pero se puede encontrar en una amplia gama de aplicaciones de software en la actualidad.*

**Metodología arquitectónica**: la arquitectura SOA.

La Arquitectura Orientada a Servicios (SOA) es un paradigma de diseño, se ha convertido en un enfoque fundamental teniendo como objetivo principal facilitar la creación de sistemas flexibles, interoperables y fáciles de mantener, al organizar la funcionalidad en servicios independientes que pueden ser consumidos por diferentes aplicaciones y componentes dentro de un ecosistema.

.

**Metodología arquitectónica**: la arquitectura SOA.

Principios Fundamentales de SOA

- **Servicios Independientes**: En SOA, la funcionalidad se descompone en servicios individuales. Estos servicios son unidades lógicas de trabajo que pueden ser desarrolladas, probadas y desplegadas de forma independiente.
- **Interoperabilidad**: Los servicios en una arquitectura SOA deben ser capaces de comunicarse entre sí, independientemente de la tecnología o el lenguaje de programación en el que estén implementados. Los estándares como XML, JSON y protocolos como HTTP se utilizan comúnmente para lograr esta interoperabilidad.
- **Reutilización**: Uno de los principales beneficios de SOA es la capacidad de reutilizar servicios en diferentes partes de una organización o en diferentes proyectos.
- **Descubrimiento y Registro**: Los servicios en una arquitectura SOA deben estar registrados en un repositorio central para que puedan ser descubiertos y utilizados por otros componentes del sistema que los necesiten.
- **Seguridad y Gestión**: La seguridad y la gestión de servicios son aspectos críticos en SOA.

**Metodología arquitectónica**: la arquitectura SOA.

Componentes de SOA

- **Servicios**: Son las unidades fundamentales en SOA. Representan una pieza de funcionalidad autónoma y se comunican a través de interfaces estandarizadas.
- **Registro de Servicios**: Un repositorio central donde se registran y se pueden buscar los servicios disponibles en la arquitectura SOA.
- **Orquestación y Coreografía**: Mecanismos para coordinar y controlar la secuencia de ejecución de servicios. La orquestación se enfoca en el control centralizado, mientras que la coreografía se enfoca en la coordinación distribuida.
- **Mediadores y Transformadores**: Componentes que se utilizan para enrutar, transformar y traducir mensajes entre diferentes servicios.

**Metodología arquitectónica**: la arquitectura SOA.

Beneficios de SOA en Ingeniería

- **Flexibilidad**: Permite cambios y actualizaciones más sencillos debido al modularidad de los servicios.
- **Reutilización**: Facilita la reutilización de servicios en diferentes proyectos, lo que reduce los costos de desarrollo.
- **Interoperabilidad**: Permite la integración de sistemas heterogéneos y la comunicación entre aplicaciones de diferentes tecnologías.
- **Escalabilidad**: Los sistemas basados en SOA pueden escalar de manera efectiva al agregar o quitar servicios según sea necesario.
- **Mantenibilidad**: La separación de servicios facilita la identificación y corrección de problemas, lo que reduce los tiempos de inactividad.

**Metodología arquitectónica**: la arquitectura SOA.

Desafíos de SOA

- **Complejidad**: La gestión de un gran número de servicios puede ser compleja y requerir un enfoque sólido de gobierno de servicios.
- **Seguridad**: La exposición de servicios puede ser un riesgo de seguridad si no se implementan las medidas adecuadas.
- **Gestión de Ciclo de Vida**: La gestión de versiones y la evolución de los servicios a lo largo del tiempo.

**Metodología arquitectónica**: la arquitectura Microservicios.

Los microservicios son una arquitectura que ha revolucionado la forma para diseñar, desarrollar y mantener aplicaciones.

Estudios de Casos:

- Netflix: La transición de una arquitectura monolítica a microservicios.
- Amazon: AWS y su enfoque en microservicios.

**Metodología arquitectónica**: la arquitectura Microservicios.

Los microservicios en el contexto de la ingeniería de datos se refieren a una arquitectura que se utiliza para diseñar y desarrollar sistemas de procesamiento de datos distribuidos y escalables.

Al igual que en otras áreas de la informática, los microservicios en ingeniería de datos se basan en el principio de descomponer una aplicación o sistema en componentes independientes y pequeños, llamados microservicios, que pueden ser desarrollados, implementados y escalados de manera independiente.

**Metodología arquitectónica**: la arquitectura Microservicios.

Características:

- **Independencia**: Cada microservicio en una arquitectura de microservicios de datos se enfoca en una tarea específica, como la ingestión de datos, el procesamiento, el almacenamiento o la presentación de resultados. Esto permite que los equipos trabajen de manera independiente en cada microservicio sin afectar a los demás.
- **Escalabilidad**: Los microservicios se pueden escalar de manera independiente según la demanda. Esto significa que puede asignar más recursos (como servidores o contenedores) a un microservicio particular si necesita aumentar su capacidad de procesamiento de datos, sin afectar a los demás microservicios.
- **Tecnología diversa**: Cada microservicio puede utilizar diferentes tecnologías o lenguajes de programación según lo que mejor se adapte a su tarea específica. Esto permite la flexibilidad en la elección de herramientas y tecnologías adecuadas para cada componente.
- **Comunicación entre microservicios**: Los microservicios se comunican entre sí a través de interfaces bien definidas, como API REST o mensajes, lo que facilita la integración y la coordinación entre ellos.
- **Despliegue y mantenimiento independiente**: Puede implementar y actualizar cada microservicio de manera independiente, lo que facilita la implementación continua y la entrega rápida de nuevas funcionalidades.
- **Resiliencia y tolerancia a fallos**: La arquitectura de microservicios permite diseñar sistemas más resilientes, ya que un fallo en un microservicio no necesariamente afecta a todo el sistema. Los errores pueden ser aislados y manejados de manera más efectiva.
- **Gestión de datos distribuidos**: Los microservicios en ingeniería de datos a menudo se utilizan para gestionar grandes volúmenes de datos distribuidos, incluida la ingesta de datos desde múltiples fuentes, su procesamiento y almacenamiento, y la entrega de resultados a los usuarios finales.

**Metodología arquitectónica**: la arquitectura Microservicios.

Implementación de Microservicios:

- Diseño de servicios independientes.
- Comunicación entre microservicios.
- Implementación de API Gateway.
- Uso de contenedores (con Docker y/o Podman) y orquestación (como Kubernetes).

**Metodología arquitectónica**: la arquitectura Microservicios.

Desafíos de los Microservicios:

- Gestión de la complejidad.
- Coordinación entre servicios.
- Monitoreo y depuración.
- Seguridad y autenticación.

**Herramientas:**

- **UML (Unified Modeling Language):** Un lenguaje de modelado estándar que incluye una variedad de diagramas para representar la arquitectura.
- **Aplicaciones de modelado**: Herramientas como Enterprise Architect, Visual Paradigm y Lucidchart permiten crear diagramas arquitectónicos.
- **Plataformas de diseño de microservicios**: Herramientas como Kubernetes, Podman, Docker.

**Desafíos:**

- Cambios en los requisitos.
- Complejidad.
- Alineación con objetivos del proyecto.

**Evolución:**

- Con la adopción de enfoques ágiles, el diseño arquitectónico tiende a ser más iterativo y adaptable.
- La nube y la computación distribuida han influido en la evolución de las arquitecturas, llevando a arquitecturas escalables y resilientes.

**Plataformas de diseño de microservicios**: Herramientas como Kubernetes, Podman, Docker.

Introducción a la contenerización

La contenerización es una forma de virtualización del sistema operativo en la que se ejecutan aplicaciones monolíticas y/o microservicios [4], en espacios de usuario aislados llamados contenedores, en un entorno informático portable, empaquetado, aislado, con los insumos necesarios para que una aplicación requiera ejecutar (binarios, bibliotecas, dependencias, archivos de configuración), permitiendo la interoperabilidad de grupos de trabajo basados en DevSecOps (Developer – Security - Operations) [1], que tienen procesos y procedimientos de flujos en integración y entrega continua (Pipeline CI/CD) [2], en un ambiente controlado con el sistema operativo base implementado en Bare Metal, On-Premises y/o Cloud Computing [3].

**Plataformas de diseño de microservicios**: Herramientas como Kubernetes, Podman, Docker.

Ventajas y beneficios

- **Portabilidad**: Un contenedor con servicios y/o microservicios, puede ser generado, transferido, y ejecutado en diferentes ambientes front and back end (desarrollo, preproducción, producción), en la plataforma transversal Institucional.
- **Velocidad**: Los equipos de desarrolladores pueden generar servicios y/o microservicios en contenedores reduciendo el tiempo de la implementación en los ambientes propuestos por la plataforma transversal, así como también pueden reutilizarse para otros proyectos gracias al acceso a registro de imágenes base.
- **Escalabilidad**: La tecnología del contenedor permite el crecimiento vertical, mediante reconfiguraciones automatizadas en la administración de recursos de hardware, dentro de la plataforma transversal, para los diferentes ambientes, con base al ciclo de vida del proyecto.
- **Agilidad**: El motor para ejecutar contenedores (Runtimes) [5], para diferentes sistemas operativos para monousuario o multiusuario bajo la administración de la iniciativa Open Container Initiative (OCI) [6], permitiendo que los equipos de desarrolladores puedan utilizar herramientas y procesos en DevSecOps.
- **Eficiencia**: Las aplicaciones ejecutadas en entornos en contenedores comparte el kernel del sistema operativo del host Bare Metal, los desarrolladores pueden compartir los servicios y/o microservicios con tiempos de respuesta agiles para su implementación en los diferentes ambientes de la plataforma transversal, reduciendo los costos de operación, y licenciamiento.
- **Aislamiento de eventualidades**: La contenerización permite aislar los procesos con base a la lógica del servicio y/o microservicio, por lo que si ocurre una excepción en tiempo de ejecución el contenedor con eventualidades puede ser removido sin afectar a los otros contenedores que estén en el front and back end, para reducir riesgos en la operación en los diferentes ambientes.
- **Privacidad y seguridad**: La independencia de los contenedores evita que el código malicioso afecte a otros servicios y/o microservicios permitiendo niveles de permisos a nivel del sistema operativo para bloquear acceso de forma automática a componentes o comandos no deseados.
- **Facilidad de administración**: Con una plataforma transversal para la orquestación de los contenedores dentro del Instituto, puede automatizar la instalación, administración, escalabilidad, registro de versiones, depuración por ciclo de vida terminado, en la operación de servicios y/o microservicios en contenedores.
- **Continuidad de operaciones**: La plataforma transversal de contenerización permite que en caso de contingencia puedan transferirse manual o automáticamente los contenedores front and backend que tengan actividades de misión crítica.

**Plataformas de diseño de microservicios**: Herramientas como Kubernetes, Podman, Docker.

Ambiente de implementación

Con base en las mejores prácticas del Cloud Native Computing Foundation (CNCF) [7], es pertinente generar una plataforma transversal orientada a la implementación de la Container Runtime Interface (CRI) [8], para que utilice instancias y entornos en tiempo de ejecución [Runtimes & Engines: (runC, crun y Kata containers)] avalados por la Open Container Initiative (OCI), considerando las siguientes características:

- **Entorno de almacenamiento (Storage Interface):** La plataforma transversal de contenerización deberá estar conectada a los esquemas de almacenamiento SAN (Storage Area Network), NAS (Network Attached Storage) [9], para que los contenedores generen volúmenes virtuales a físicos, con base al ciclo de vida del proyecto.
- **Entornos en tiempo de ejecución (Runtime):** La plataforma transversal de contenerización deberá estar conectada a la estrategia de utilizar herramienta para los entornos de desarrollo con técnicas basadas en DevSecOps
- **Entorno en conectividad (Networking Interface):** La plataforma transversal de contenerización deberá tener un direccionamiento de red de tipo interno y externo para el acceso a los servicios y/o microservicios basados en contenedores, así como también utilizar un dominio con balanceo global y certificado de seguridad de tipo "wildcard", para la transferencia de datos [10].
- **Entorno de registro (Registry):** La plataforma transversal de contenerización deberá tener un repositorio que permite tener el registro de las imágenes base que utilizan los contenedores para habilitar servicios y/o microservicios en el flujo automatizado con base al ciclo de vida del proyecto [11].
- **Privacidad**: La plataforma transversal de contenerización deberá tener un entorno de integridad, confidencialidad, disponibilidad de los contenedores generados para los ambientes de desarrollo, preproducción y producción con base al ciclo de vida del proyecto [12].

**Plataformas de diseño de microservicios**: Herramientas como Kubernetes, Podman, Docker.

Nota: una de la estrategia con mayor madurez para orquestar arquitecturas basadas en contenerización es CRI-O [13], el grupo conformado para la especificación, herramientas y control de versiones está en la referencia [14].

**Plataformas de diseño de microservicios**: Herramientas como Kubernetes, Podman, Docker.

**Orquestación de contenedores con base a PODS [Almacenamiento, Procesamiento, Transferencia]**

- Lugar [On-premises - Cloud]
- Aplicación [Monolítico, Microservicios]
- Lenguaje [Java, .NET, Python entre otros]
- Sistema Operativo [Kernel Linux]
- Implementación a escala
- Programación de cargas de trabajo
- Supervisión de estado
- Conmutación cuando se produce una eventualidad en un nodo
- Escalado o reducción vertical
- Funciones de red
- Detección de servicios
- Coordinación de las actualizaciones de aplicaciones
- Afinidad de nodos de clúster
- Seguridad [Disponibilidad, Confidencialidad, Integridad]

**Plataformas de diseño de microservicios**: Herramientas como Kubernetes, Podman, Docker.

**Seguridad de imágenes de contenedores**: estas capacidades se integran directamente con los entornos de desarrollo existentes, lo que ayuda a garantizar que las imágenes de contenedores comiencen su vida útil de acuerdo con las mejores prácticas de diseño moderno. Esto incluye escanear en busca de vulnerabilidades conocidas o de día cero en las imágenes, evitar que sean infectadas por malware, no permitir que las credenciales codificadas se filtren en ellas, etc. Los resultados de los escaneos de vulnerabilidad de contenedores deben alinearse y clasificarse de acuerdo con los modelos de evaluación de riesgos.

**Plataformas de diseño de microservicios**: Herramientas como Kubernetes, Podman, Docker.

**Seguridad del registro de contenedores**: proporciona visibilidad, control de acceso y seguridad continuos para las imágenes de contenedor almacenadas en los registros, lo que garantiza que las imágenes válidas no se vean comprometidas y se evite el acceso no autorizado, las modificaciones de imágenes o la infiltración de contenedores no autorizados.

**Plataformas de diseño de microservicios**: Herramientas como Kubernetes, Podman, Docker.

**Seguridad de la plataforma de orquestación**: la plataforma de orquestación de contenedores en sí debe estar protegida adecuadamente en todas las capas de su infraestructura subyacente, desde la seguridad de los sistemas host hasta la implementación de la segmentación de la red, el aislamiento de la carga de trabajo y la protección de todas las interfaces de administración. Se deben implementar tanto el endurecimiento proactivo como el monitoreo en tiempo real, junto con la administración de la configuración y el gobierno de acceso integral, haciendo cumplir los principios de segregación de funciones y privilegios mínimos.

**Plataformas de diseño de microservicios**: Herramientas como Kubernetes, Podman, Docker.

**Monitoreo de contenedores en tiempo de ejecución**: proporciona visibilidad continua en tiempo real de las actividades dentro de los contenedores en ejecución, utilizando tanto la detección basada en firmas como el análisis de comportamiento impulsado por ML para identificar amenazas en tiempo de ejecución. Las plataformas de seguridad de contenedores deben utilizar toda la gama de controles de seguridad en los niveles de host, red, contenedor y aplicación para bloquear o mitigar las amenazas detectadas de forma rápida y automática.

**Plataformas de diseño de microservicios**: Herramientas como Kubernetes, Podman, Docker.

**Gestión de incidentes:** estas capacidades permiten a los analistas de seguridad reaccionar rápidamente a las amenazas identificadas, realizar investigaciones forenses, tomar las decisiones correctas y, finalmente, automatizar la remediación de amenazas utilizando una combinación de controles de orquestación nativos y herramientas de seguridad especializadas.

**Plataformas de diseño de microservicios**: Herramientas como Kubernetes, Podman, Docker.

**Auditoría y cumplimiento**: el cumplimiento normativo es un desafío importante y, al mismo tiempo, un impulsor del negocio para organizaciones de cualquier tamaño o industria. La retención de datos de seguridad y los informes de cumplimiento completos son las capacidades básicas aquí. El soporte listo para usar para marcos regulatorios como GDPR, HIPAA o PCI es un diferenciador importante para muchos clientes.

**Plataformas de diseño de microservicios**: Herramientas como Kubernetes, Podman, Docker.

**Integraciones**: las soluciones de seguridad de contenedores no pueden funcionar como herramientas independientes sin integraciones profundas con servicios en la nube existentes, plataformas de orquestación de contenedores, canalizaciones DevOps y DevSecOps, así como plataformas SIEM y otras herramientas de operaciones de seguridad. Mantener un ecosistema abierto de 3Rd Las integraciones de partes son un diferenciador clave para los proveedores.

Podman:

Es una herramienta de administración de contenedores que se utiliza para crear, administrar y ejecutar contenedores en sistemas Linux. A menudo se compara con Docker, otra herramienta popular de administración de contenedores. Sin embargo, Podman tiene algunas características únicas que lo distinguen de Docker y lo hacen especialmente atractivo para ciertos casos de uso y entornos.

Origen y Motivación:

Podman es un proyecto de código abierto desarrollado por Red Hat y la comunidad de código abierto. Fue diseñado para abordar las limitaciones de Docker, especialmente en entornos donde se necesitaba un enfoque más seguro y compatible con Kubernetes.

Características Clave:

- Daemonless: A diferencia de Docker, Podman no requiere un daemon (proceso en segundo plano) para ejecutar contenedores. Esto mejora la seguridad y facilita la administración de contenedores.
- Compatibilidad con Docker: Podman es compatible con la mayoría de los comandos de Docker, lo que facilita la transición desde Docker a Podman.
- Pods: Los pods son una característica exclusiva de Podman que permite agrupar varios contenedores y compartir el mismo espacio de red y volúmenes. Esto es útil para aplicaciones que necesitan comunicarse entre sí de manera eficiente.
- Rootless: Podman permite a los usuarios ejecutar contenedores sin privilegios de root, lo que mejora la seguridad.
- CRI-O Integration: Podman se integra con CRI-O, un proyecto de contenedores OCI (Open Container Initiative) que es una parte fundamental de Kubernetes. Esto facilita la ejecución de contenedores en entornos Kubernetes.

Ventajas:

Seguridad Mejorada: El enfoque "sin daemon" y la capacidad de ejecutar contenedores sin privilegios de root hacen que Podman sea más seguro en comparación con Docker.

Compatibilidad con Docker: Los usuarios que están familiarizados con Docker pueden cambiar a Podman sin una curva de aprendizaje significativa.

Pods: La capacidad de crear pods es útil para aplicaciones que requieren múltiples contenedores trabajando juntos.

Universidad de la Ciudad de Aguascalientes

Casos de Uso:

Podman es adecuado para una variedad de casos de uso, incluyendo el desarrollo de aplicaciones, la implementación de contenedores en servidores y la orquestación de contenedores en Kubernetes.

Comunidad y Soporte:

Podman cuenta con una comunidad activa de desarrolladores y usuarios que brindan soporte y contribuyen al proyecto. Además, Red Hat ofrece soporte comercial para empresas que utilizan Podman en entornos empresariales.

https://podman-desktop.io/

Ejemplo de Uso: crear y ejecutar un contenedor con Podman:

**podman run -it --rm ubuntu:20.04 /bin/bash**

Referencias


[1] O. Diaz, M. Muñoz and J. Mejia, "Responsive infrastructure with cybersecurity for automated high availability DevSecOps processes," 2019 8th International Conference On Software Process Improvement (CIMPS), 2019, pp. 1-9, doi: 10.1109/CIMPS49236.2019.9082439.
[2] Khushalani, Prateek. (2022). CI/CD Systems. 10.1007/978-1-4842-8032-4_3.
[3] Tfrifonov, Daniel & Valchanov, Hristo. (2018). VIRTUALIZATION AND CONTAINERIZATION SYSTEMS FOR BIG DATA.
[4] Jha, Devki Nandan & Garg, Saurabh & Jayaraman, Prem Prakash & Buyya, Rajkumar & Li, Zheng (Eddie) & Morgan, Graham & Ranjan, R.. (2019). A study on the evaluation of HPC microservices in containerized environment. Concurrency and Computation: Practice and Experience. 33. 10.1002/cpe.5323.
[5] Wang, Xingyu & Du, Junzhao & Liu, Hui. (2022). Performance and isolation analysis of RunC, gVisor and Kata Containers runtimes. Cluster Computing. 25. 1-17. 10.1007/s10586-021-03517-8.
[6] https://opencontainers.org/community/overview/
[7] https://www.cncf.io/about/members/
[8] https://github.com/kubernetes/kubernetes/blob/242a97307b34076d5d8f5bbeb154fa4d97c9ef1d/docs/devel/container-runtime-interface.md
[9] (1) Gandhi, Arun & Varki, Elizabeth & Bhatia, Swapnil. (2002). Reader-Writer Locks for Network Attached Storage and Storage Area Networks.. 402-405. (2) N. Zhao et al., "Large-Scale Analysis of Docker Images and Performance Implications for Container Storage Systems," in IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 4, pp. 918-930, 1 April 2021, doi: 10.1109/TPDS.2020.3034517.
[10] M. M. Khalel, M. Arul Pugazhendhi and G. R. Raj, "Enhanced Load Balancing in Kubernetes Cluster By Minikube," 2022 International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN), 2022, pp. 1-5, doi: 10.1109/ICSTSN53084.2022.9761317.
[11] Buchanan, Steve & Rangama, Janaka & Bellavance, Ned. (2020). Container Registries. 10.1007/978-1-4842-5519-3_2.
[12] Colman, Matt. (2022). Containers and Kubernetes: Security is not an Afterthought. ITNOW. 64. 44-45. 10.1093/itnow/bwac023.
[13] https://cri-o.io/
[14] https://github.com/opencontainers

2. Diseño

2.1 Diseño de procesos propuestos.
2.1.1 Herramientas CASE para diseño.
2.2 Diseño arquitectónico.
**2.3 Diseño de datos.**
2.4 Diseño de interfaz de usuario.

**Definición:**

El diseño de datos se refiere al proceso de planificar y organizar cómo se capturarán, almacenarán, procesarán y gestionarán los datos en un sistema.

Esto incluye la definición de estructuras de datos, esquemas de bases de datos, modelos de datos, políticas de seguridad y estrategias de acceso a los datos.

**Importancia:**

- **Eficiencia**: Un diseño de datos sólido puede mejorar el rendimiento de una aplicación, permitiendo una recuperación inmediata de la información.
- **Integridad de los datos**: Ayuda a garantizar que los datos se almacenen de manera precisa y consistente.
- **Escalabilidad**: Un diseño adecuado de datos facilita la expansión del sistema a medida que crece la cantidad de información.
- **Seguridad**: Contribuye a proteger los datos sensibles y a garantizar el cumplimiento de regulaciones de privacidad.
- **Interoperabilidad**: Facilita la integración con otros sistemas y aplicaciones.

**Elementos clave:**

- **Modelo de datos**: Define la estructura lógica de los datos, incluyendo tablas, relaciones y atributos en el caso de bases de datos relacionales.
- **Esquema de bases de datos**: Describe la organización física de los datos en el almacenamiento, como índices, particiones y fragmentación.
- **Políticas de seguridad**: Establece quién puede acceder a los datos y qué acciones pueden realizar.
- **Optimización de consultas**: Considera cómo se accederán y procesarán los datos de manera eficiente.

**Métodos y enfoques:**

- **Modelado de datos**: Utilización de técnicas como el Modelo Entidad-Relación (ER), el Modelo Relacional y el Modelo Dimensional para representar la estructura de los datos.
- **Normalización**: Proceso de organizar los datos en tablas para reducir la redundancia y mejorar la integridad de los datos.
- **Denormalización**: Estrategia que se utiliza en ocasiones para mejorar el rendimiento al permitir cierta redundancia controlada.
- **Gestión de versiones de datos**: Implementación de estrategias para controlar y gestionar las versiones de los datos considerando su ciclo de vida.

**Herramientas:**

- **Sistemas de gestión de bases de datos (DBMS)**: Herramientas como Oracle DB, MySQL, Microsoft SQL Server y PostgreSQL proporcionan capacidades de diseño y administración de bases de datos.
- **Herramientas de modelado de datos**: Ejemplos incluyen Erwin, IBM Data Architect y MySQL Workbench.
- **Herramientas de ETL (Extract, Transform, Load)**: Utilizadas para transformar y cargar datos desde diversas fuentes a sistemas de almacenamiento.

**Desafíos:**

- **Cambios en los requisitos.**
- **Mantenimiento**: La gestión continua de la integridad y la calidad de los datos puede ser un desafío a lo largo del tiempo.
- **Escalabilidad**: Diseñar para manejar grandes volúmenes de datos.

## Evolución:

- Con la creciente adopción de arquitecturas de microservicios y sistemas distribuidos, el diseño de datos debe considerar la descentralización y la replicación de datos.
- El auge del "big & fast data", ha llevado a la exploración de nuevas tecnologías y enfoques, como bases de datos NoSQL y sistemas de almacenamiento distribuido.