Terminales

"Internet de las Cosas"

Centros de Datos

"Aplicaciones de Microservicios"

Anatomía de una Factoría de Aplicaciones

-DevSecOps-

ÍNDICE

1.- EL RETO DE LA SOBERANÍA DIGITAL

1.1.- LA SOCIEDAD DE LA INFORMACIÓN.

Por doquier apreciamos cómo las aplicaciones informáticas infiltran cada intersticio de nuestras vidas cotidianas. En el coche guían nuestra ruta, en el móvil nos mantienen conectados a toda hora, en el trabajo preparando informes u organizando la agenda de eventos. Ningún espacio, ni ninguna actividad se escapa a su influjo, un fenómeno que hemos acordado llamar "la transformación digital".

Sin darnos cuenta, somos cada vez más dependientes de un mundo virtual del que poco sabemos. Desde las corporaciones más grandes hasta cada uno de nosotros -a título individual-, nadie es capaz de sustraerse a esta poderosa influencia… estamos todos a merced de estas aplicaciones informáticas, cual boyas a la deriva de corrientes oceánicas.

¿A dónde nos arrastran las corrientes de este océano digital? El cine recurrentemente evoca el miedo a ser dominados por unas máquinas que nosotros mismos hemos creado. Sentimos que esa nueva realidad nos arrastra a un futuro incierto, sacrificando el tesoro de nuestra intimidad por el camino.

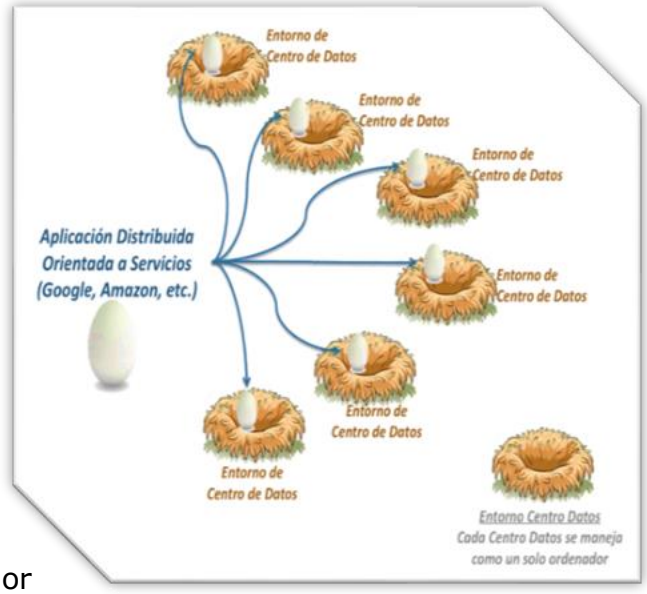
Sin embargo, todo este universo virtual que nos envuelve se reduce a aplicaciones informáticas: en todo su amplio espectro de objetivos y tecnologías de construcción… desde sistemas de control de vuelo, pasando por la robótica de las fábricas, inteligencia artificial, big data, hasta las aplicaciones de nuestros ordenadores personales o teléfonos móviles.

En el ADN encontramos codificados los esquemas organizativos de todos los sistemas que forman un organismo vivo. La información existe para organizar y coordinar la vida, es decir, para el orden y buen compás que rige este Cosmos. Las sociedades humanas no escapan a esta ley, por eso nos vemos rodeados de todas esas herramientas informáticas que sirven para organizarnos mejor como sociedad, en eso que rotulan como "sociedad de la información". Se torna vital, pues, no dejar la producción de esas aplicaciones que organizan todas nuestras vidas en manos de miopes designios comerciales, que nos atrapan en experiencias virtuales con inciertas intenciones y dudosa utilidad.

Le invitamos a acompañarnos en este viaje al corazón de la artesanía del software, desvelar los secretos de su concepción y manufactura para saltar las cerraduras de este mundo virtual respondiendo a la pregunta, como sociedad, ¿qué queremos hacer con todas esas aplicaciones informáticas? Nociones básicas para tomar las riendas de esta nueva economía digital, cada cual desde su frente de batalla: juristas, políticos, ingenieros y usuarios.

Se busca, pues, respuesta a las preguntas: **¿Cómo es el sistema de producción que suministra aplicaciones a la sociedad? ¿Qué agentes y responsabilidades tiene? ¿Qué medios se necesitan para poder fabricar aplicaciones seguras y estables?**

1.2.- LAS APLICACIONES DE INTERNET.



Las aplicaciones de internet son un tipo especial de aplicaciones: aplicaciones orientadas a servicios[1], aplicaciones que se ejecutan sobre racimos de ordenadores (en lugar de uno solo), para así poder soportar cualquier carga de usuarios que se conecte a la vez. Se despliegan en centros de datos, o sea, agrupaciones de muchísimos ordenadores, pero que operan todos ellos al unísono, como si de un solo ordenador se tratase[2].

En *"Teoría Pura del Capital"*, Friedrich von Hayek sostiene que la riqueza de las naciones radica en cadenas de valor mejor estructuradas. Gracias a unos medios de producción más diferenciados en cada eslabón de la cadena, se es capaz de suministrar bienes y servicios de mayor calidad de manera mucho más fluida, mejorando las condiciones de vida a una población en constante crecimiento.

Para lograr este prodigioso aumento de productividad, la sociedad debe sacrificar sus ahorros (su capital), durante todo el tiempo que se requiera hasta disponer de esos medios de producción que necesitan las cadenas de valor. En otras palabras, sacrificar el consumo inmediato en favor de una mayor productividad futura. Por ejemplo, distinto es trabajar con un arado romano de dos bueyes, que con un tractor. Sin embargo, la producción de tractores es un proceso mucho más complejo, que insume buena parte del capital de la sociedad hasta poder contar con todos y cada uno de los medios con los que poder fabricar esos tractores

Aplicando esta teoría pura del capital al caso de la computación, obtenemos que **la normalización del sistema de producción de aplicaciones orientadas a servicios posibilita crear y suministrar unos medios de producción que garanticen un abastecimiento fluido, diverso y rico de este tipo de aplicaciones** para así tomar las riendas de este sistema de producción y, por ende, de esta sociedad de la información. Se trata de circunscribir fines y medios en estas cadenas de valor:

- Los Fines, ¿Qué? - Normalizar las responsabilidades de cada eslabón de esta cadena para garantizar aplicaciones estables y seguras.
- Los Medios, ¿Cómo? - Normalizar los medios de producción que necesita cada eslabón para cumplir con esas responsabilidades, garantizado un suministro diferenciado en constante evolución que permitan surgir un efervescente ecosistema productivo.

---

[1] *Universidad de Bradford, 'Integración de servicios aeronáuticos basados en SOA'* https://www.researchgate.net/publication/221917207_SOA-Based_Aeronautical_Service_Integration

[2] ***Jimmy Clidaras*** *(arquitecto de centros de datos de Google), "**El centro de datos como un solo computador**" (Morgan and Claypool publishers):* https://www.morganclaypool.com/doi/10.2200/S00874ED3V01Y201809CAC046

1.3.- LA CADENA DE SUMINISTRO DE APLICACIONES.

Toda cadena de valor es un sistema distribuido, por lo tanto, consta de tres eslabones: producción, distribución y acceso. En la imagen se aprecian las responsabilidades en el caso de la computación para lograr suministrar aplicaciones estables y seguras a la sociedad:

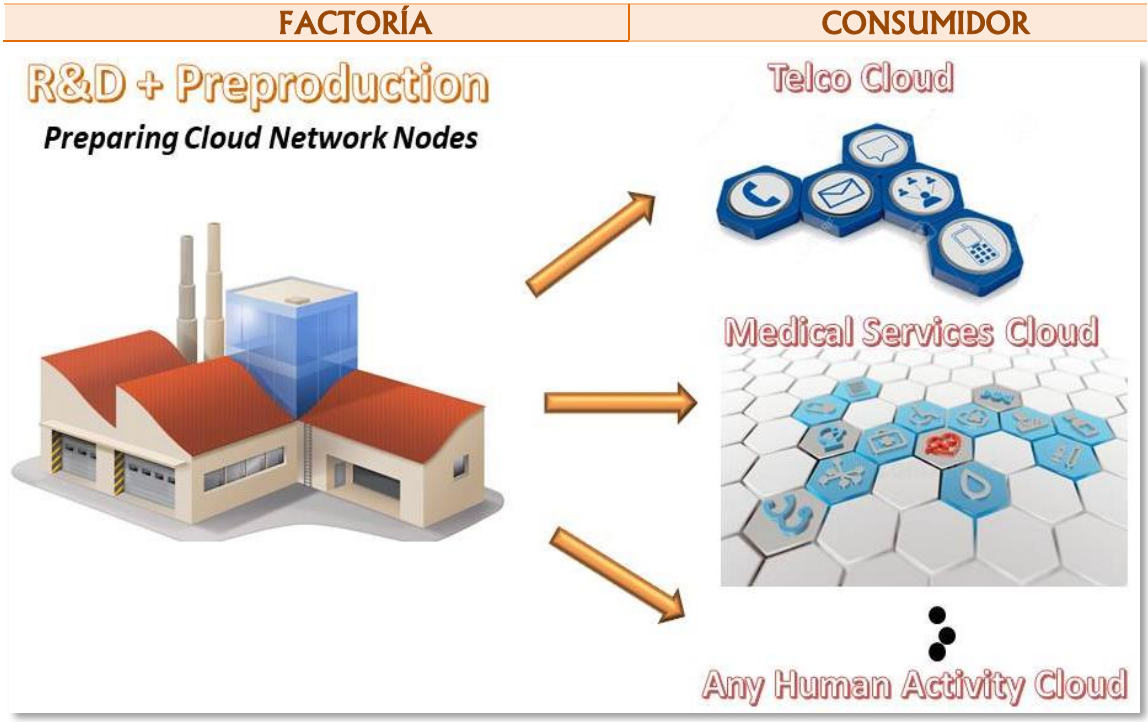
- **Acceso, despliegue de aplicaciones**: plataforma de despliegue de servicios y sistema de identidad para acceder al ecosistema de aplicaciones. Todo parece apuntar que se emplearán tarjetas SIM asociadas a cada dispositivo, obviando el sistema actual de contraseñas.
- **Distribución, homologación de servicios**: garantizar las condiciones de despliegue de los servicios que, cual piezas de lego, se emplean para componer aplicaciones finales… siendo suministrados y actualizados, de manera continua, a través de un sistema de repositorios.
- **Producción, factorías de aplicaciones**: diseñar aplicaciones bajo metodología DevSecOps que garantiza parámetros de estabilidad y seguridad. Esto implica:
    - *Controlar la exposición de la superficie de datos* a través de políticas de acceso.
    - *Visualizar el sistema de dependencias* entre servicios, para mantener estables los contratos de funcionalidades que ofrece cada servicio.
    - *Gestionar el correcto encapsulado de los servicios* en contendores para su posterior distribución.

1.4.- DIGITAL MARKETS ACT: EL PROBLEMA DEL ACCESO A LOS MEDIOS DE PRODUCCIÓN.

Europa es dependiente de operadoras de centros de datos extranjeros[3], como podrían ser Amazon, Google o Azure... esto quiere decir, que sectores como la banca digital o las administraciones del Estado no pueden costearse su propia maquinaria, tienen que alquilar a operadores de centros de datos, originando el problema de la soberanía digital europea[4]. Digital Markets Act intenta paliar el impacto[5]. En otras palabras, al no tener acceso a sus propios medios de producción y despliegue de aplicaciones orientadas a servicios, se ven abocados a alquilarlos, originando el problema de la soberanía digital.

¿Cuáles son los medios de producción y despliegue de esas aplicaciones orientadas a servicios? Normalizarlos para suministrarlos[6], pequeños y baratos, resuelve este dilema de la soberanía digital… tan vital en unas economías cada vez más dependientes de internet.

Respecto a la maquinaria, cada ubicación donde van a ejecutarse aplicaciones deberá levantar una plataforma de ejecución de servicios y un repositorio de contenedores homologados (cada contenedor encapsula un servicio con su entorno de ejecución). Respecto a las herramientas, cada factoría tiene que localizar la suite de aplicaciones que le permita automatizar las dinámicas DevSecOps hasta hacerlas lo más transparentes que sea posible a los programadores de aplicaciones, simplificándoles su trabajo diario.



[3] *Foro Económico Mundial, 'El 92% de todos los datos del mundo occidental se alojan en servidores propiedad de empresas con sede en Estados Unidos:* https://www.weforum.org/agenda/2021/03/europe-digital-sovereignty/

[4] *Financier Worldwide Magazine, El Desafía de esta Época: La Soberanía Digital y Europa:* https://www.financierworldwide.com/epochal-struggle-digital-sovereignty-and-the-eu

[5] *Wikipedia, Digital Markets Act:* https://en.wikipedia.org/wiki/Digital_Markets_Act

[6] *Nokia Data-Center Delivery Service:* https://youtu.be/nCKNIYdp7_Y

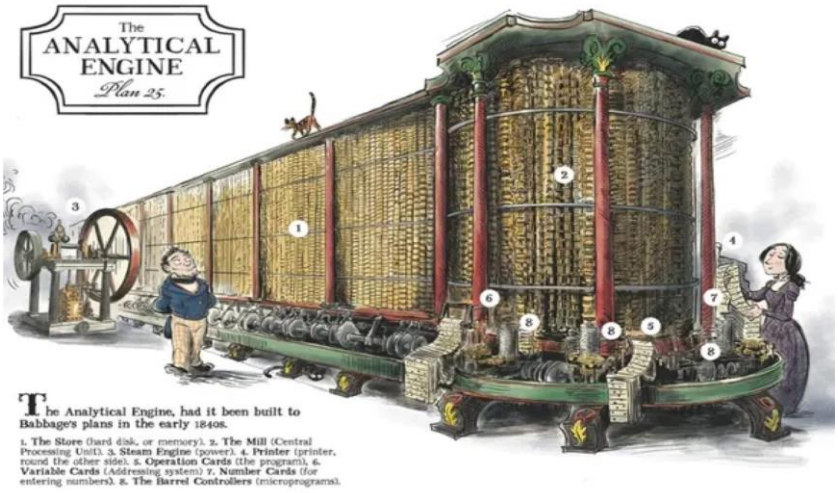
1.5.- UN ACCESO LIMITADO A LOS MEDIOS: EL MODELO DE PRODUCCIÓN ARTESANAL.

Imaginemos que tuviéramos que gestionar nuestro portátil como hacemos en esos centros de datos. Cada vez que abriésemos un navegador o un editor de textos... manualmente hay que localizar en qué CPU y memoria van a ejecutarse, qué caché está disponible en cada momento y cómo va a liberarse, etc. El coste de administrar todas las aplicaciones de un escritorio de ordenador sería prohibitivamente alto... estaríamos obligados a delegar todo ese trabajo manual en agentes especializados que administren las aplicaciones; y serían entornos compartidos con muchos otros usuarios.

Esto es lo que sucede con los centros de datos[7], unos pocos proveedores de nube han logrado una infraestructura capaz de desplegar muchas de esas aplicaciones; el resto de fabricantes de aplicaciones, ora alquila la maquinaria en la que poder desarrollar su trabajo, ora procura artesanalmente armar su propia plataforma que a duras penas es capaz de gestionar un reducidísimo número de aplicaciones, por el alto coste que esto implica.

No se ha llegado a normalizar la estructura de la pieza[8] para que proliferen más operadores de centros de datos, así que dependemos de unos pocos de estos operadores que concentran toda nuestra información, dando pie al problema de la soberanía digital.

Afortunadamente, al contrario de lo que sucede con esos centros de datos[9], nuestros portátiles tienen un sistema operativo que nos abstrae de todo ese trabajo manual de asignación de recursos, y así cada persona puede tener su propio PC. De hecho, esa maquinaria nos recuerda mucho a aquellos primeros ordenadores donde había que empeñar un ejército de operarios para ejecutar el más simple de los programas.



The Analytical Engine, had it been built to Babbage's plans in the early 1840s.

1. The Store (hard disk, or memory). 2. The Mill (Central Processing Unit). 3. Steam Engine (power). 4. Printer (printer, round the other side). 5. Operation Cards (the program). 6. Variable Cards (Addressing system) 7. Number Cards (for entering numbers). 8. The Barrel Controllers (microprograms).

[7] *Benjamin Hindman, cofundador del proyecto D2IQ, ¿por qué el centro de datos necesita un sistema operativo?:* https://www.oreilly.com/content/why-the-data-center-needs-an-operating-system/

[8] *Universidad de Stanford, "el oscuro arte de la construcción de nubes de computación":* http://web.stanford.edu/class/cs349d/

[9] *Universidad de Berkeley, '**El centro de datos necesita un sistema operativo**':* https://people.eecs.berkeley.edu/~alig/papers/datacenter-needs-os.pdf

1.6.- UN SUMINISTRO DIFERENCIADO DE MEDIOS: EL MODELO DE PRODUCCIÓN INDUSTRIAL.

Igual que tenemos redes de telefonía móvil a nivel global; estas nubes son redes de centros de datos que permiten sincronizar un tejido social en tiempo real y a nivel mundial… juzgados, policías, ejércitos, etc. No dependen de las geografías (como las redes de comunicaciones), sino de la actividad económica que han de administrar: misma actividad, mismas aplicaciones.

Esto quiere decir que el contenido de estas redes de centros de datos son aplicaciones destinadas a aliviar las tareas administrativas de las distintas actividades económicas. Pero producir este tipo de aplicaciones demanda factorías software, con sus sofisticados procesos de producción, y los correspondientes medios de fabricación.

El qué deben hacer esas aplicaciones para aliviar la carga administrativa, compete a los organismos normalizadores de cada actividad económica. Sin embargo, el sistema de producción de aplicaciones orientadas a servicios es el mismo en todo escenario, siendo crucial normalizarlo para originar un rico y diverso ecosistema de factorías capaz de adaptarse al vastísimo espectro de necesidades según cada circunstancia específica.

La única referencia de normalización del sistema de producción de aplicaciones es el sistema de recomendaciones de fabricación establecidas por el Departamento de Defensa de Estados Unidos [DoD 1,2] para todas sus factorías software. **Surge como respuesta a los problemas de ciberseguridad, ya que una producción industrial garantiza aplicaciones seguras de fábrica** (véase sección 3.2.2), en lugar de depender de la definición que haga cada centro de datos para el esquema de zonas de su perímetro de seguridad. Los problemas de ciberseguridad que vayan surgiendo son llevados a instalaciones especializadas[10] en determinar sus causas, para reportarlas a las factorías de aplicaciones, que introducirán los correctivos necesarios en sus procesos de fabricación (en un ciclo de ininterrumpido de mejora).

[10] *Nokia Berlin Security Center: https://youtu.be/JIEoRChIus8*

1.7.- NORMALIZACIÓN DEL SISTEMA DE PRODUCCIÓN.

La normalización del sistema de producción descrito eventualmente pueda ser procurada alineando intereses en una misma dirección:

- **Platform One** *[DoD_1,2]*: una infraestructura única y común para todo el desarrollo software del Departamento de Defensa de Estados Unidos.
  - *Valor:* la capacidad de suministrar recomendaciones de fabricación que puedan transformarse en estándares para la industria[11].
  - *Necesidades:* buscan ampliar su fuerza de trabajo.
- **ETSI – European Telecommunication Standards Institute** *[ETSI_18]*: instituto que ha producido especificaciones tan importantes como GSM o 5G.
  - *Valor:* suministrar recomendaciones de fabricación para redes de centros de datos (a imagen y semejanza de las redes de telefonía móvil), que reduzcan la inversión necesaria para establecer nuevas factorías software… originando un nuevo tejido industrial que *proporciona las aplicaciones que requieren este tipo de redes*.
  - *Necesidades:* diversificación del sector para amortizar las enormes inversiones iniciales que tiene toda operadora de telecomunicaciones.
- **GAIA-X**: proyecto que aspira a una infraestructura digital europea.
  - *Valor:* financiación pública europea[12].
  - *Necesidades:* modelo de gobierno que alinee toda su fuerza de trabajo sin el cual los fondos europeos no obtienen resultados comercializables, esto es, retorno a la inversión[13].

Parte esencial de la normalización del sistema de producción es determinar cuáles son los medios de producción y despliegue de aplicaciones que necesita esta metodología de trabajo, simplificando y automatizando la burocracia que coordina todos los procesos de fabricación de una factoría. Se reducen los costes, tanto de establecer nuevas factorías, como de modernizar las antiguas para adaptarlas a los nuevos estándares de ciberseguridad. La única manera de organizar un tejido industrial destinado a aliviar las tareas administrativas de todas las actividades económicas.

El resultado es un ecosistema productivo totalmente integrado, en lugar del actual fragmentado, al provenir de un modelo "start-up" que carece de una finalidad, así que ni obtiene resultados finales ni establece ninguna senda evolutiva. En contraste, tenemos el tejido de fabricantes que produce todas y cada una de las piezas que componen las redes de telecomunicaciones… un tejido que trabaja para resolver los requisitos de las operadoras de comunicaciones que despliegan las redes (requisitos canalizados a través de instituciones de normalización[14]).

---

[11] *Platform One, introducción y búsqueda de colaboradores: https://youtu.be/yUOjL7G9mIg*

[12] *Thierry Breton, coordinación europea para resolver su problema de soberanía digital: https://ec.europa.eu/commission/presscorner/detail/en/IP_22_3898*

[13] *Martin Bayer, "dudas sobre el futuro del proyecto GAIAX": https://www.computerworld.es/tecnologia/gaiax-fracasara-la-nube-de-datos-europea*

[14] *Francisco Javier Ramón, ETSI Open Source MANO Chair, "la coordinación global del tejido industrial que produce Telco Clouds": https://youtu.be/51EKdEnbvHc?t=989*

1.8.- ESTRUCTURA DEL DOCUMENTO.

Este documento toma como referencia el sistema de recomendaciones establecidas por el Departamento de Defensa de Estados Unidos para todas sus factorías software (recomendaciones públicas y accesibles desde internet *[DoD_1,2]*), a fin de revisar sus medios de producción: maquinaria y herramientas. Es decir, *aquellos recursos indispensables en toda factoría software que quiera aplicar metodología DevSecOps en sus procesos de fabricación*, ajustándose así a estándares modernos de seguridad informática.

Se comienza con una visión de conjunto del mundo de la tecnología y el papel que juega la nube en todo esto[15]. Se prosigue revisando contenido y continente de esas nubes: ecosistemas de aplicaciones orientadas a servicios y plataforma de entrega continua. Se finaliza viendo los diferentes contextos donde se despliegan esas nubes y el estado del arte:

- **La era de la nube**: ¿qué es una nube? ¿por qué es el corazón mismo de la sociedad de la información?
- **SOA - Services Oriented Architecture**: la nube como medio para poder fabricar y desplegar aplicaciones orientadas a servicios *[Erl_4]*.
- **Plataforma de entrega continua - El entorno de ejecución de servicios**: el uso de una única plataforma (tanto por usuarios de aplicaciones como factorías software) permite establecer una metodología de trabajo común. El uso de un repositorio de distribución de servicios homologados, garantiza consistencia en las aplicaciones entregadas por las factorías. El resultado de combinar ambas cosas (calidad en los métodos de producción y distribución; y consistencia en las aplicaciones suministradas y distribuidas), se traduce en: mayor productividad, mejor calidad y menor coste de esas aplicaciones.
- **Los entornos de trabajo - Consumidores y factorías de aplicaciones**: esta sección revisa los diferentes casos de uso de la combinación plataforma y repositorio vista en la sección anterior.
- **Estado del arte**: fabricantes y consumidores de aplicaciones solo pueden optar por modelos de suscripción:
  - Maquinaria externa: alquilar maquinaria a operadores de centros de datos que tienen una plataforma de grandes capacidades.
  - Maquinaria propia: suscribirse a servicios que automaticen su propia maquinaria (de un lado, la plataforma, con agentes como D2IQ, del otro cada una de las aplicaciones que deba ser ejecutada).

[15] *Departamento Defensa de Estados Unidos, DAU (Defense Acquisition University):* ***What is DevSecOps and Why Does DoD Care?:*** https://media.dau.edu/media/t/1_oh7drwzn

2.- LA ERA DE LA NUBE.

2.1.- EL CONTEXTO.

2.1.1.- EL CENTRO DE DATOS: UNA AGRUPACIÓN DE ORDENADORES.

Un centro de datos es un grupo de ordenadores que trabajan al unísono, como si fueran un solo ordenador, y al que se conectan todos los usuarios de un hospital o de un juzgado, para así tener centralizada y controlada su información de negocio. Cuenta con un sistema de credenciales para cada usuario, al margen del terminal con el que quiera conectarse en cada lugar y en cada momento (su contexto de conexión *[NIST_3]*).

En estos centros de datos se instalan aplicaciones orientadas a servicios que, en lugar de ejecutarse sobre un solo ordenador, se ejecutan sobre racimos de ordenadores (o clústeres en inglés) para poder soportar cualquier carga de usuarios que se conecte a la vez.

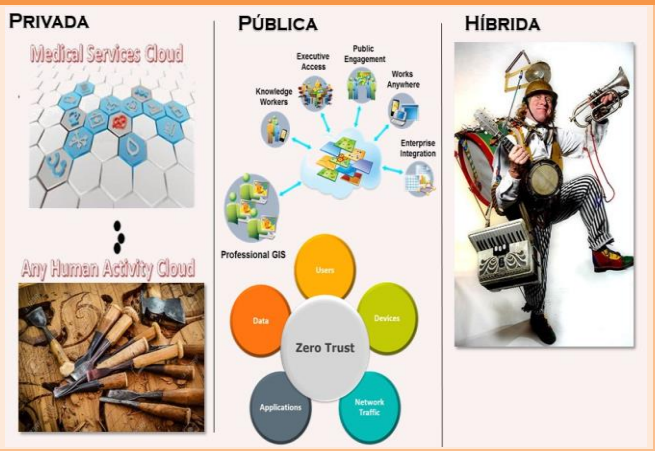
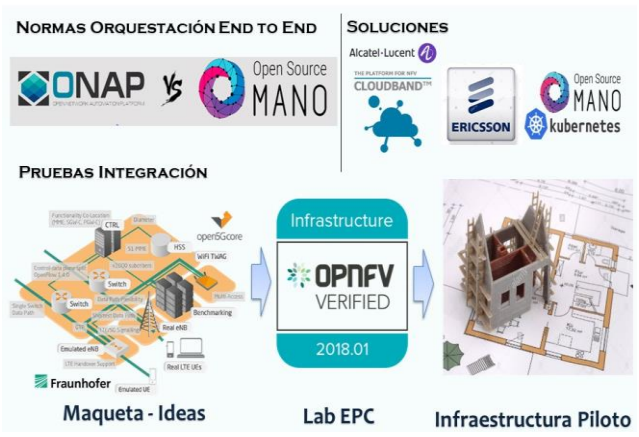
2.1.2.- LA NUBE: UNA FEDERACIÓN DE CENTROS DE DATOS.

Cuando se conectan varios hospitales entre sí, es decir, cuando **se federan sus centros de datos se forma una nube, una red de centros de datos que coordina una actividad a lo largo de toda una región, o de todo un país**: juzgados, hospitales, supermercados, bancos, etc.

Estas serían nubes de ámbito privado, en el ámbito público encontramos los servicios de internet -como Google, Amazon, Facebook, etc.- que, además de las aplicaciones que todos conocemos, alquilan maquinaria o aplicaciones a empresas (en sus distintas modalidades comerciales: IaaS, PaaS, CaaS, NaaS, SaaS que se explicarán en la sección 2.4).

2.1.3.- TIPOS DE NUBES: TELCO Y COMPUTING CLOUDS.

Existen dos familias de nubes diferentes, las aplicaciones se alojan en nubes de computación (*Computing Clouds*), pero se accede a esas aplicaciones a través de redes de comunicaciones que están migrando a nubes de telecomunicaciones (*Telco Clouds*).

TIPO NUBE	OBJETIVO	DESCRIPCIÓN
Computing Cloud Aplicaciones	• Privado: tareas administrativas de Empresas e Instituciones. • Público: aplicaciones de Internet. • Híbrido: datos críticos gestionados en privado, datos no esenciales gestionados en nubes públicas.	
Telco Cloud Núcleos de Red 5G	• Privado: internet de las cosas. • Público: ciudades inteligentes.	

2.1.4.- KUBERNETES (K8S): STANDARD DE FACTO PARA GESTIONAR RACIMOS DE ORDENADORES

Kubernetes es un estándar de facto para la gestión de racimos de ordenadores (o clústeres en inglés) donde desplegar aplicaciones orientadas a servicios.

Los servicios son piezas software preconstruidas, que conforman un catálogo de componentes básicas reutilizables, que pueden combinarse para crear cualquier aplicación final. En la electrónica, existe un catálogo de circuitos integrados con los que crear cualquier dispositivo electrónico para automoción, electrodomésticos, calderas, equipos de música, etc. Los servicios vienen a ser esto mismo, pero para la producción de aplicaciones software.

El diseño de aplicaciones orientadas a servicios surge para reducir el coste de mantenimiento de los centros de datos, por las dos ventajas inherentes a la granularidad de los servicios:

- **Reutilización de la lógica**: mismas piezas de lego, que se combinan de distintas maneras para crear cualquier aplicación final, aumenta la fiabilidad, reduce la cantidad de código a mantener y evita la acumulación de aplicaciones obsoletas (el uso de interfaces estandarizadas permite tanto reescribir piezas completas con tecnologías más modernas, como recombinar las mismas piezas en nuevas soluciones).
- **Crecimiento en los datos**: la fragmentación de la superficie de datos posibilita reacondicionarla de manera más sencilla, es decir, adaptar los esquemas de crecimiento de los datos: reduciendo costes de mantenimiento (al evitar grandes migraciones de datos) y aumentando su visibilidad (reduciendo riesgos).

2.1.5.- LA IMPORTANCIA DEL PROCESAMIENTO PARALELO.

Para poder desplegar esos servicios es necesario ser capaces de gestionar masivamente terminaciones de lógica donde alojarlos, y esto tiene dos requisitos primarios:

- **Suministro Software**: a partir del cual aprovisionar los entornos de computación donde viven esos servicios. Las distribuciones Linux (distros) son el estándar de facto para las líneas de aprovisionamiento software, donde destacan las ramas rpm y deb.

- **Procesamiento Paralelo**: capaz de gestionar masivamente el ciclo de vida de esas terminaciones de lógica, de manera muy similar a como un sistema operativo gestiona los procesos en un ordenador personal. En la actualidad, se emplean racimos de ordenadores Linux gestionados con Kubernetes, en lugar de maquinaria multiprocesador.

Es precisamente la incapacidad de lograr procesamiento paralelo fácil de usar y accesible a todo el mundo, lo que convierte a los propietarios de estas capacidades en monopolios tecnológicos. Aunque existan ya proyectos de investigación en la línea de suministrar esas capacidades a un mayor porcentaje de fabricantes (como los Sistemas Operativos Amoeba, Sprite o el sistema Constellation System de Sun Microsystems)… están muy lejos de poder ser comercializables. Es necesario resolver muchos retos de investigación.

A continuación, se expondrá cómo se diseñan esas aplicaciones orientadas a servicios para ver cuán importante es el procesamiento paralelo en todo esto… al margen de la unidad computación que quiera emplearse (convencional o cuántica).

2.2.- LA INDUSTRIALIZACIÓN DEL SOFTWARE.

2.2.1.- TRANSICIÓN A UNA FILOSOFÍA ORIENTADA A SERVICIOS.

La imagen representa la evolución de las técnicas de desarrollo software en los últimos años *[DoD_1,2]*. Esencialmente consiste en la **industrialización de la producción del software gracias a una programación orientada a servicios**, que reduce tiempos de entrega a la vez que mejora la calidad del producto final.

Cada uno de estos servicios (de esas componentes software preconstruidas) implementa una función muy específica, en contraste con las soluciones monolíticas "todo-en-uno". Una fabricación especializada de cada pieza, con sus fases de pruebas individualizadas, aumenta mucho la fiabilidad de la componente y por ende de las aplicaciones finales que las vayan a utilizar. Además, los entornos de ejecución modernos son capaces, tanto de lanzar más o menos réplicas de cada servicio al son del tráfico de peticiones entrantes, como detectar y regenerar réplicas de servicio que estén fallando.

En suma, las arquitecturas orientadas a servicios aumentan mucho la fiabilidad, escalabilidad y elasticidad de las aplicaciones, convirtiéndose en una filosofía de diseño óptima para la lógica de servidor. De un lado, un diseño de servicios basado en interfaces estandarizadas elimina la dependencia con las tecnologías de fabricación, es decir, se puede reconstruir las mismas piezas con tecnologías más modernas siempre que se respeten las interfaces. Del otro lado, la granularidad de los servicios evita grandes migraciones de datos. El resultado de combinar granularidad y estandarización, es la reducción de costes en el mantenimiento y evolución de los centros de datos.

El precio que se paga es una mayor complejidad, tanto en el diseño de las aplicaciones (con su sistema de dependencias entre servicios *[Erl_23]*) como su despliegue en sofisticados entornos de ejecución de servicios.

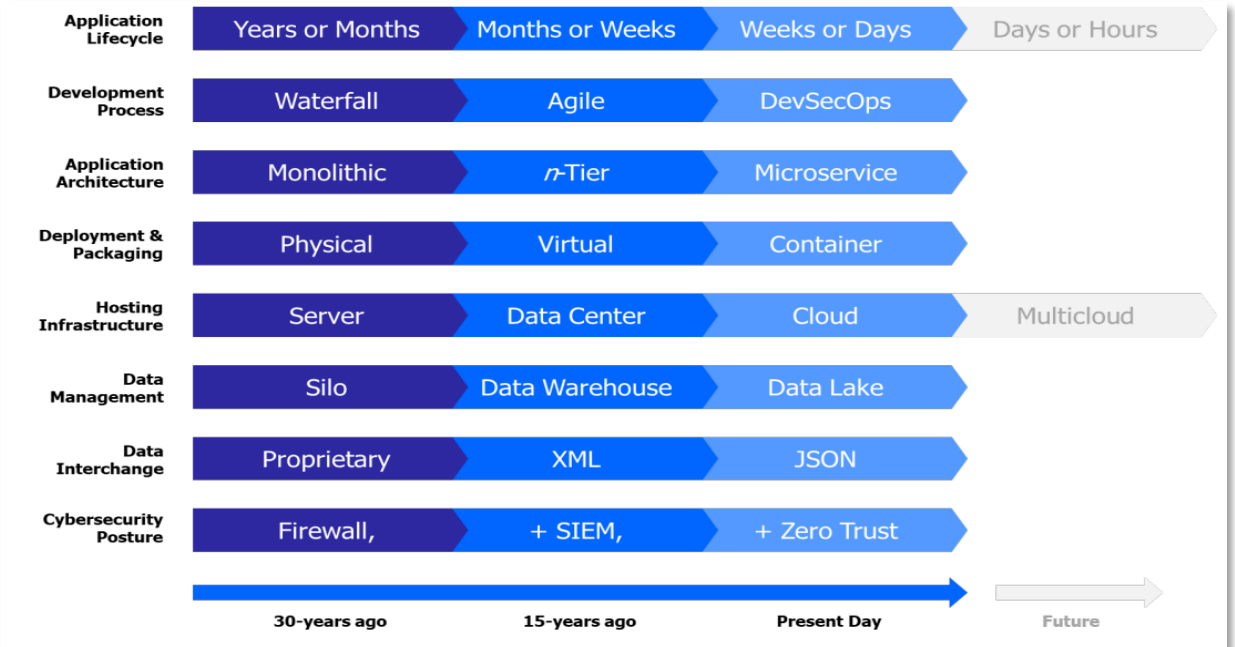


Figure 3 Maturation of Software Development Best Practices

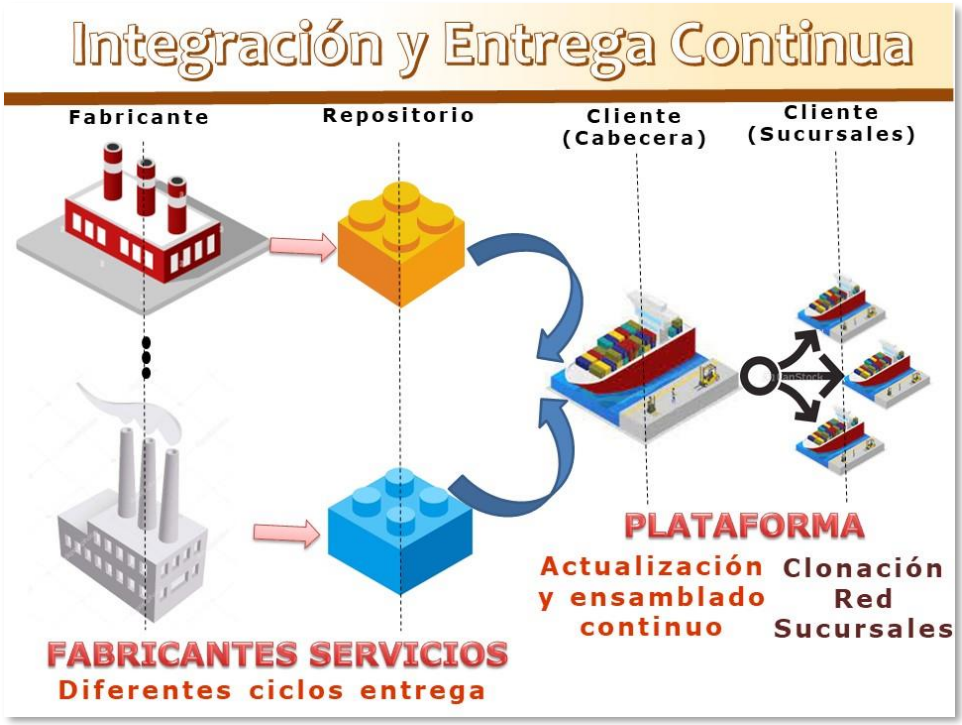
2.2.2.- EL SISTEMA DE ENTREGA CONTINUA.

Es necesario encapsular esos servicios en entornos aislados (terminaciones de lógica), y levantar un sistema que permita su ensamblado para crear cualquier aplicación final, es decir, un entorno de ejecución (o Run-Time) de servicios.

Debido a la naturaleza del software, esos servicios han de actualizarse de manera continua en cada una de las aplicaciones que lo usen. Los servicios se van actualizando en la medida que cada fabricante los vaya liberando a un repositorio de servicios (que hace las veces de un Apple Store o Play Store en el caso de aplicaciones de teléfonos móviles).

Nuestra plataforma deberá desplegar las nuevas versiones de los servicios (a partir de los repositorios) y garantizar que las aplicaciones siguen funcionando según lo esperado tras cada actualización… para esto se emplean pruebas de integración automatizadas. Las aplicaciones monolíticas son tratadas como una aplicación que solo tiene un servicio, pero se sigue desplegando a partir del repositorio de contenedores como el resto de aplicaciones.

El resultado final, es una cascada ininterrumpida de actualizaciones sobre todas las aplicaciones del ecosistema, ya que un mismo servicio puede formar parte de muchas aplicaciones a la vez. Sin embargo, en la práctica, ha de existir algún alineamiento en los ciclos de entrega de esos servicios, es muy difícil lograr un auténtico sistema de entrega continuo.



La producción artesanal del software probablemente quede relegada a las aplicaciones que van en los terminales de usuario (SmartPhones, Tablets, etc.), adoptando esta filosofía más industrial para la lógica crítica de negocio que reside en centros de datos.

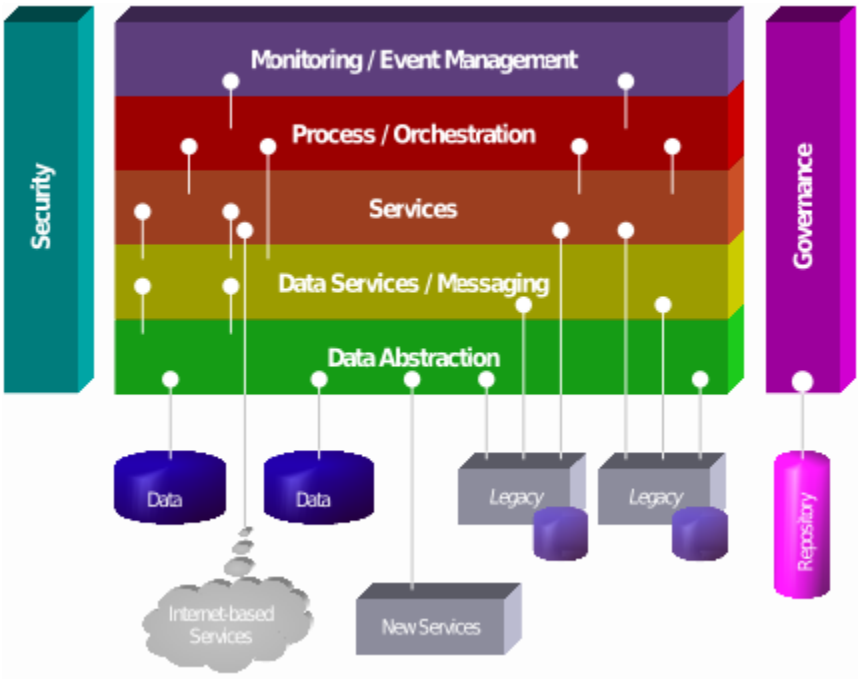
3.- SOA: SERVICES ORIENTED ARCHITECTURE.

3.1.- LA ORIENTACIÓN A SERVICIOS.

3.1.1.- LA FILOSOFÍA: EL MODELADO DE LA REALIDAD

Las aplicaciones orientadas a servicios salen como resultado de una filosofía basada en servicios, un modo de modelar la realidad como composición de servicio. Esto es, un servicio global (como un servicio bancario) se forma como resultado de ir federando dominios de servicios (recursos humanos, contabilidad, ventas, etc.). Cada dominio está compuesto por servicios cada vez más específicos (cada persona juega un rol, es decir, ofrece un servicio) que se coordinan para formar contextos.

3.1.2.- EL META-MODELO: ORGANIZACIÓN DE LA INFRAESTRUCTURA DE SERVICIOS

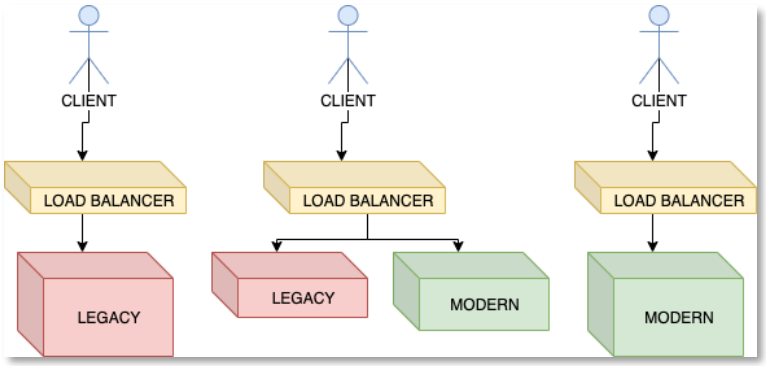


3.2.- PRINCIPIOS DE DISEÑO: LA ESTRUCTURA.

3.2.1.- EL DISEÑO DE APLICACIONES: DOMAIN DRIVEN DESIGN AND STRANGLE PATTERN.

Las aplicaciones orientadas a Servicios siguen una filosofía de diseño basada en dominios (*DDD=Domain Driven Design* *[DoD_1]*) que consiste en dividir una aplicación monolítica en dominios independientes, esto es, dominios que admiten ser implementados por diferentes equipos de desarrollo de manera totalmente autónoma, sin acoplo entre dominios. Si aparecen acoplamientos, es mejor mantener una única aplicación monolítica, por los problemas derivados de un diseño con responsabilidades compartidas por varios equipos de desarrollo, y con difícil ensamblado final, al ser muy complejo diagnosticar y resolver incidencias… llegando a ser necesaria una reorganización completa del código fuente.

La transición de una estructura monolítica a una orientada a servicios, se realiza siguiendo el *"Strangle Pattern"* que muestra la imagen *[Fowler_5]*, que permite progresivamente ir reorganizando el código de la aplicación a la par que se van incorporando nuevas funcionalidades, es decir, no detener la evolución del código fuente a raíz de su reorganización interna.



En suma, es decisivo el papel de los arquitectos de servicios, aquellos capaces de localizar el modelo de acoplamiento entre componentes: **cuál es el reparto de responsabilidades por dominio, que permita a la estructura incorporar nuevas funcionalidades de manera natural**, adaptándose sin esfuerzo a cualquier circunstancia que surja durante su maduración futura. Además, una buena arquitectura permite elegir distintas opciones tecnológicas para los distintos componentes, en función de la estrategia que se quiera seguir, a fin de lograr un determinado objetivo en el producto final (optimizar la velocidad, la persistencia, el precio, etc.).

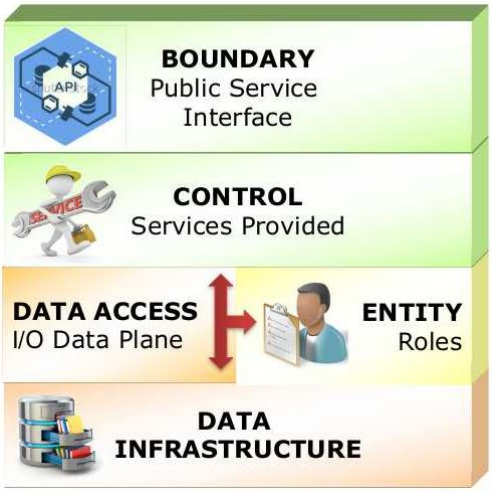
> Es de importancia crítica: a) la gestión de dependencias entre componentes (para lo que es crucial un buen diseño de APIs *[Erl_23]*); b) saber crecer en funcionalidades, con una asignación correcta de responsabilidades.

En ocasiones, para acelerar la transición a una arquitectura de servicios, se opta por un esquema organizativo que no contempla requisitos de crecimiento; eventualmente pueda desembocar en una futura parálisis del desarrollo *[Bryzek_19]*, al requerir una reorganización interna mucho más compleja cuanto más tiempo haya transcurrido.

3.2.2.- LA ESTRUCTURA DEL SERVICIO: ENTIDAD-CONTROLADOR-FRONTERA.

3.2.2.1 ESQUEMA DE ROLES: ENTITY-CONTROL-BOUNDARY PATTERN.

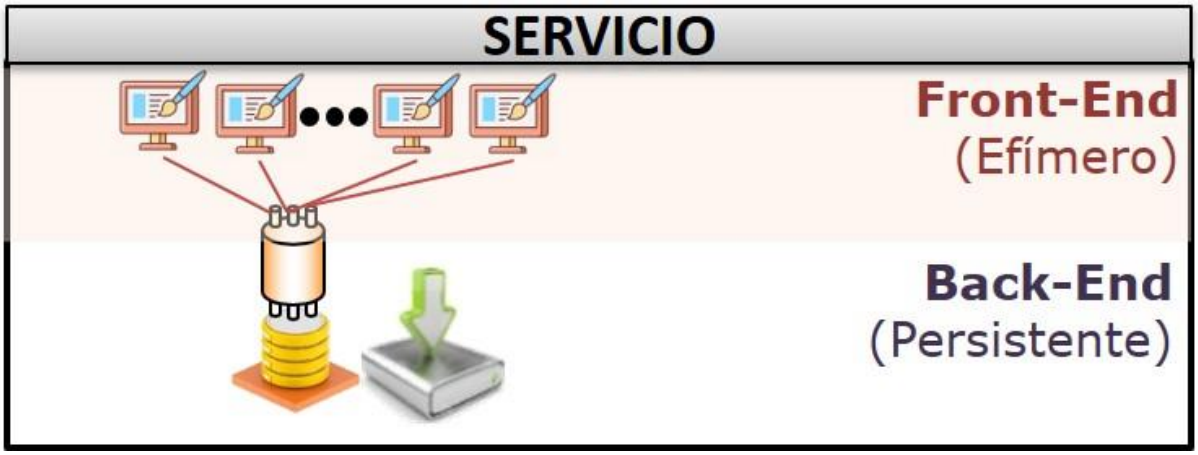
Cada dominio puede ser implementado por uno o varios servicios, cada uno de ellos empleando las mejores tecnologías a disposición para resolver su funcionalidad específica. Pero en todos los casos suelen diseñarse siguiendo el patrón arquitectónico *"Entidad-Controlador-Frontera"* que se muestra en la imagen, un esquema organizativo que facilita la interconexión con el resto de servicios en los entornos más heterogéneos *[Oracle_11]*.



CAPA	DESCRIPCIÓN	PROPIEDADES
Frontera *[Erl_23]*	**API** (Application Programming Interface)	Interfaz Pública: contrato de prestaciones que ofrece el servicio. Finalidad: encapsulado, integración. Validaciones: parametrización de las llamadas.
Controlador	**Implementación de la API**	Implementación del contrato, control del ciclo de vida del servicio. Finalidad: toma de decisiones centralizado. Validaciones: puntos de decisión del ciclo de vida.
Entidad	**Roles y responsabilidades** del servicio	Reparto de responsabilidades: las entidades deben poder traspasar la frontera gracias a mecanismos de serialización, y han de ser gestionadas mediante colecciones. Finalidad: desacoplo de funcionalidades. Validaciones: cumplimiento de responsabilidades.
Acceso al Plano de Datos	**Portabilidad de entidades**: emplear cualquier plano de datos una vez reconstruidas las entidades serializadas.	Abstraer los mecanismos de acceso a los datos: interfaz pública de llamadas de acceso a los datos, controlado desde el plano de control que asigna referencias a las entidades cuando quieren acceder a los datos. Finalidad: adaptabilidad a cualquier infraestructura de datos. Validaciones: políticas de acceso a los datos para cada llamada de API.
Infraestructura de Datos	**Persistencia:** acceso a la infraestructura de datos.	Acceso persistente a los datos: gestionar conexiones al plano de datos con todas sus eventualidades. Finalidad: minimizar disrupciones del servicio. Validaciones: tolerancias sin disrupción de servicio.

3.2.2.2 DESPLIEGUE: FE, BE, DATOS.

Los servicios se despliegan mediante la arquitectura de tres niveles que muestra la imagen:

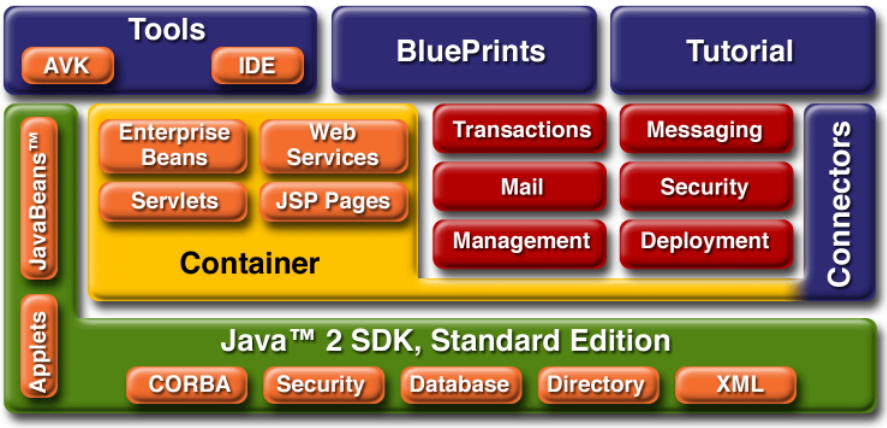


NIVEL	CAPAS SERVICIO	PROPIEDADES
Front-End	**Lógica de Servicio**	Entidad-Controlar-Frontera: gestión de datos para ofrecer funcionalidades. Finalidad: facilitar el manejo de los datos. Validaciones: escala el número de réplicas al son del volumen de peticiones entrantes.
Back-End	**Lógica de Gestión de Datos**	Esquemas de crecimiento de datos: depósito de datos por capas que permita su crecimiento. Finalidad: enrutamiento a lo largo de los contenidos gracias a una eficaz gestión de meta-datos. Validaciones: acceso persistente a la infraestructura de datos.
Infraestructura de datos	**Dispositivos de almacenamiento** del servicio	Almacenamiento Seguro de datos, con mecanismos de recuperación ante errores. Finalidad: estabilidad. Validaciones: tiempo de respuesta y recuperación ante errores.

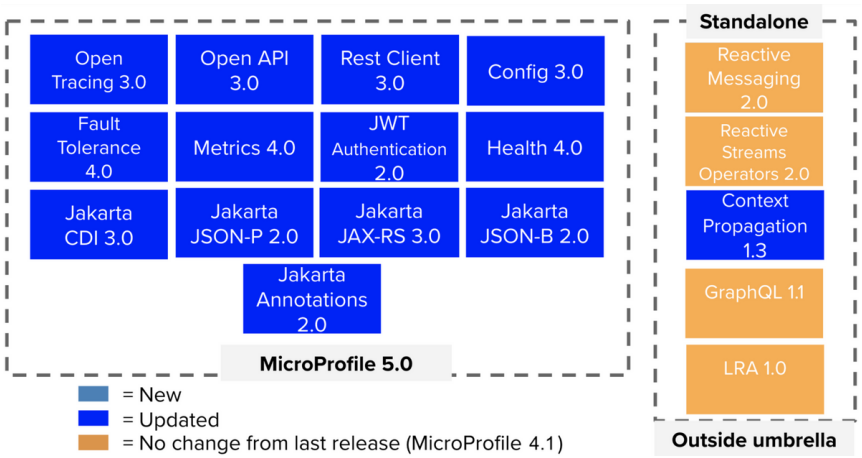
3.2.2.3 TECNOLOGÍAS: ECOSISTEMA JAVA {J2EE/MICROPROFILE-QUARKUS}.

En el año 2001 Sun Microsystems diseñó un proceso llamado *"Java Community Process"* para la creación de estándares software con los que fabricar aplicaciones empresariales con Java y centralizó muchos de los esfuerzos de llevar Java a la web en lo que se conoció como el proyecto J2EE.

Las aplicaciones empresariales de Java *[Erl_12]* son un caso particular de aplicaciones orientadas a servicios, *desplegadas en servidores de aplicaciones*. Cada una de las partes del modelo SOA que aquí se describe se traducirá en especificaciones de la norma J2EE de Java *[Eclipse_13]*, pudiendo existir diferentes especificaciones para una misma componente (por ejemplo, existen dos estándares para el manejo de la persistencia).

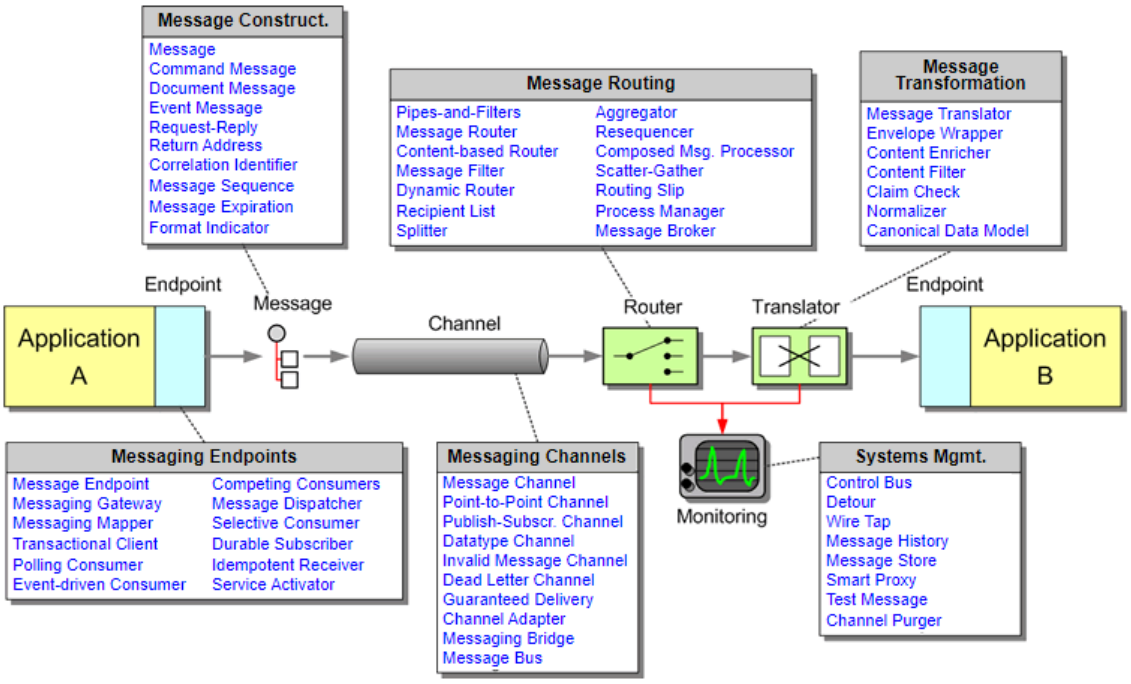


En la medida que maduran las tecnologías de nube, aparece la posibilidad de desplegar esos servicios web que utiliza J2EE (en lugar de dentro de servidores de aplicaciones) en entornos aislados, llamados contenedores, empleando HTTP como middleware ligero para las comunicaciones, surgen así los microservicios. Como respuesta a esta evolución tecnológica se crea el proyecto MicroProfile.io *[Apache_14]*, un subconjunto de especificaciones de J2EE para la construcción de microservicios. En el mundo de los contenedores, la portabilidad la da el propio contenedor, así que la máquina virtual de Java (JVM) es reemplazada por una versión mucho más rápida (GraalVM) que descarta el bytecode como lenguaje interpretado por cada procesador particular, así surgen proyectos como Quarkus: un subconjunto de especificaciones MicroProfile.io compatibles con GraalVM.

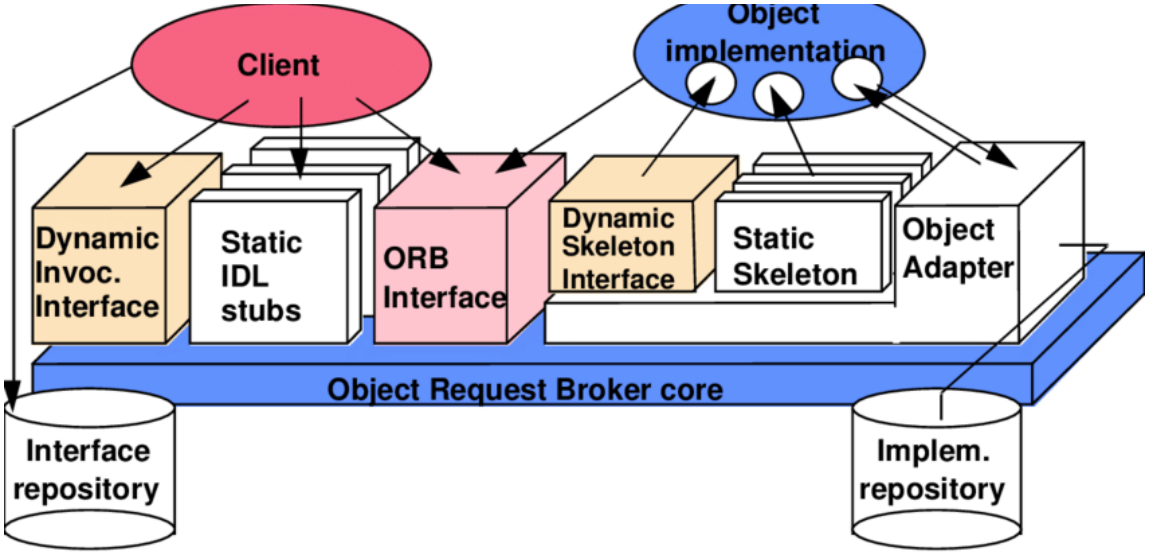


3.2.3.- COMUNICACIONES ENTRE SERVICIOS: MIDDLEWARE.

3.2.3.1 ESQUEMA DE ROLES: ENTERPRISE INTEGRATION PATTERNS.



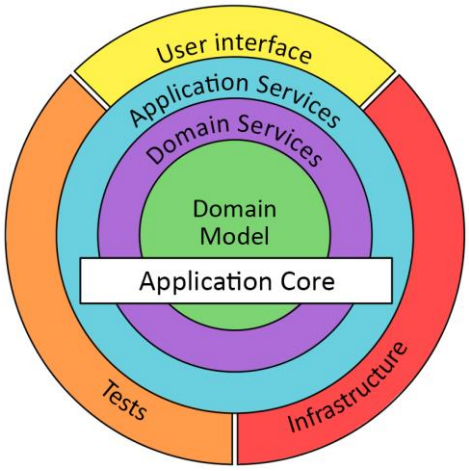
3.2.3.2 DESPLIEGUE: SISTEMAS DE MENSAJERÍA, GESTIÓN DE EVENTOS.



3.2.3.3 TECNOLOGÍAS: REST, SOAP.

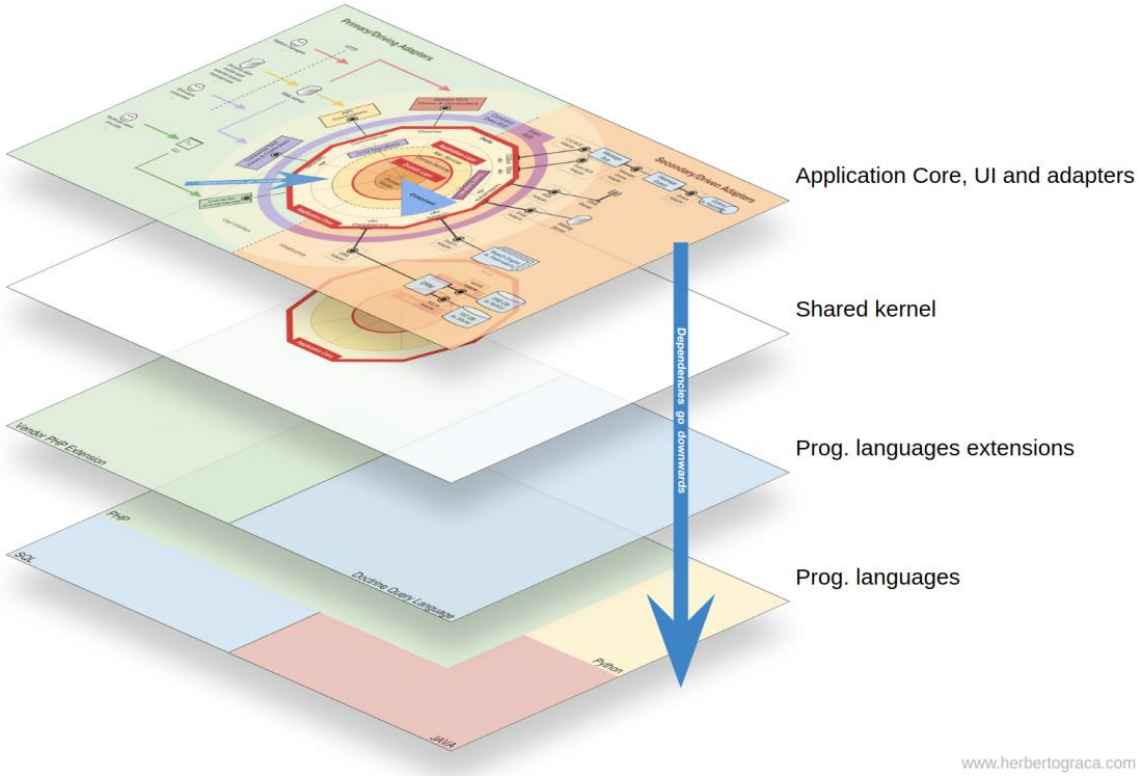
3.2.4.- MODELO DE COHESIÓN ENTRE SERVICIOS: SISTEMA DE CONTEXTOS Y ORQUESTACIÓN.

3.2.4.1 ESQUEMA DE ROLES: SERVICE LAYER PATTERN.

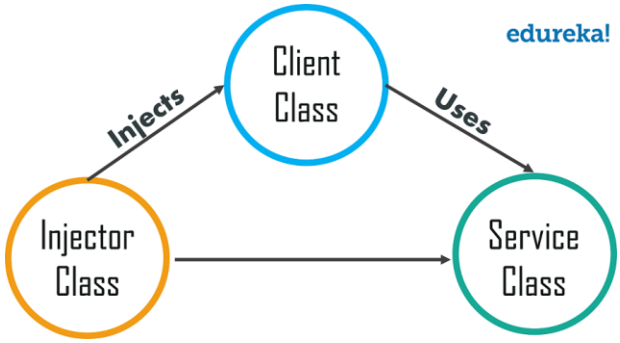


Un sistema de contextos siempre tiene como objetivo crear una estructura de capas, que descompone un sistema complejo en subsistemas independientes más simples, llamados contextos. Cada contexto o capa tiene como objetivo abstraer y encapsular toda su complejidad, ofreciendo una interfaz de manejo lo más simple que sea posible, para lograr una eficaz coordinación con el resto de capas.

Niveles bajos de acoplamiento y alta de cohesión en las componentes de cada uno de los contextos aportan la granularidad que va a permitir al sistema software crecer, esto es, ir incorporando, de manera natural, todas las nuevas funcionalidades que su incansable evolución demande.



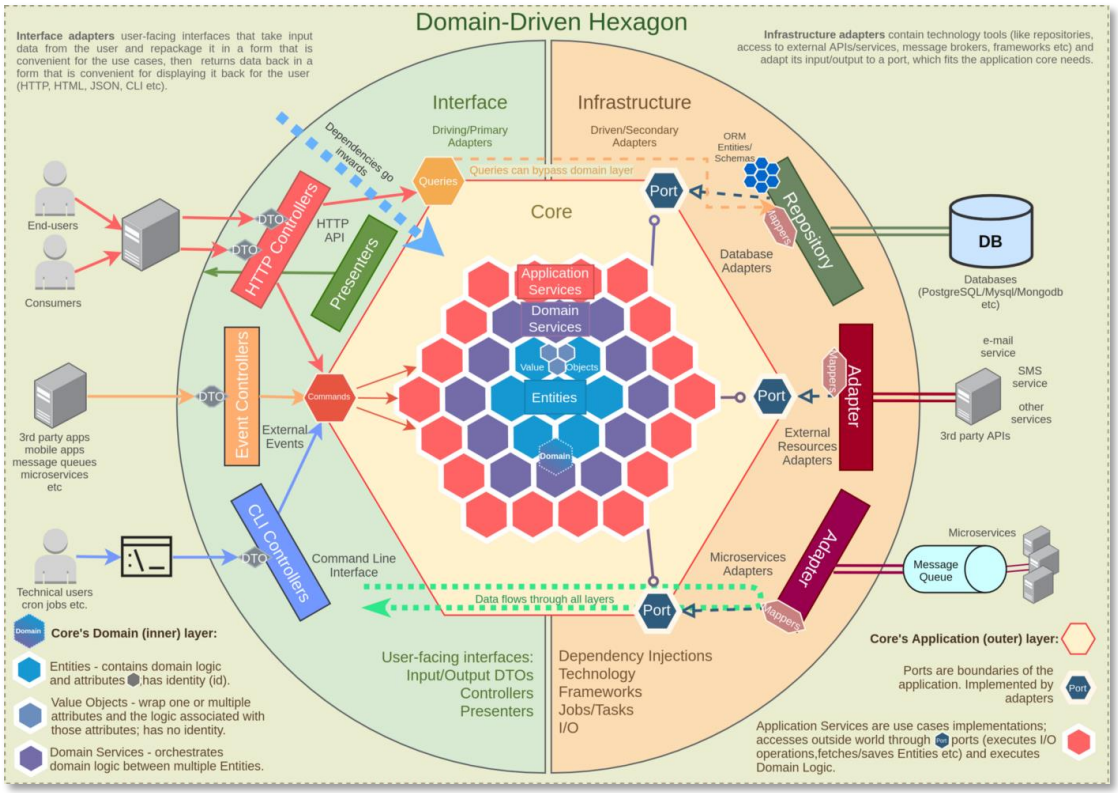
El modelo de cohesión en cada contexto depende de los patrones inyección de dependencias y localizador de servicios, que permiten a los componentes de cada capa acceder a funcionalidades compartidas sin necesidad de conocer todos sus detalles; gracias a un sistema de control por capa o inyector de dependencias.



3.2.4.2 DESPLIEGUE: ORQUESTACIÓN DE SERVICIOS.

La orquestación de servicios en cada contexto (o modelo de cohesión de la capa), tiene la siguiente estructura:

- **Interfaz – La articulación del contexto**: un registro de servicios, un orquestador (normalmente la interfaz pública de la capa) y un sistema de logging que resuma la actividad.
- **Infraestructura – La operativa del contexto**: modelo de compartición de recursos (inyección de dependencias, localizador de servicios).



3.2.4.3 TECNOLOGÍAS: BPEL, SERVIDOR APLICACIONES, SERVERLESS.

3.3.- METODOLOGÍA DE DESARROLLO: LAS DINÁMICAS.

3.3.1.- LA METODOLOGÍA: CICLO DE VIDA DEVSECOPS.

El conjunto de prácticas para el desarrollo de este tipo de aplicaciones orientadas a servicios recibe el nombre de DevSecOps (representados en la imagen inferior [DoD_1]), una evolución de la metodología Agile cuya finalidad es agilizar el desarrollo software a la par de mejora su calidad.

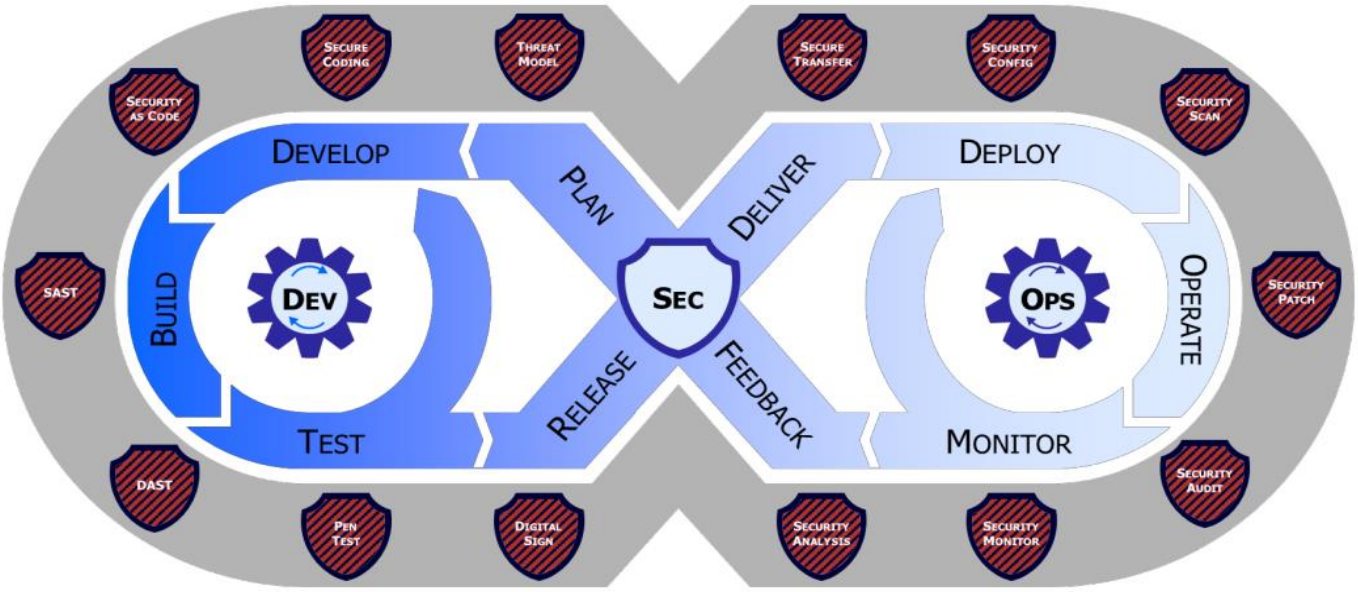


Figure 4 DevSecOps Distinct Lifecycle Phases and Philosophies

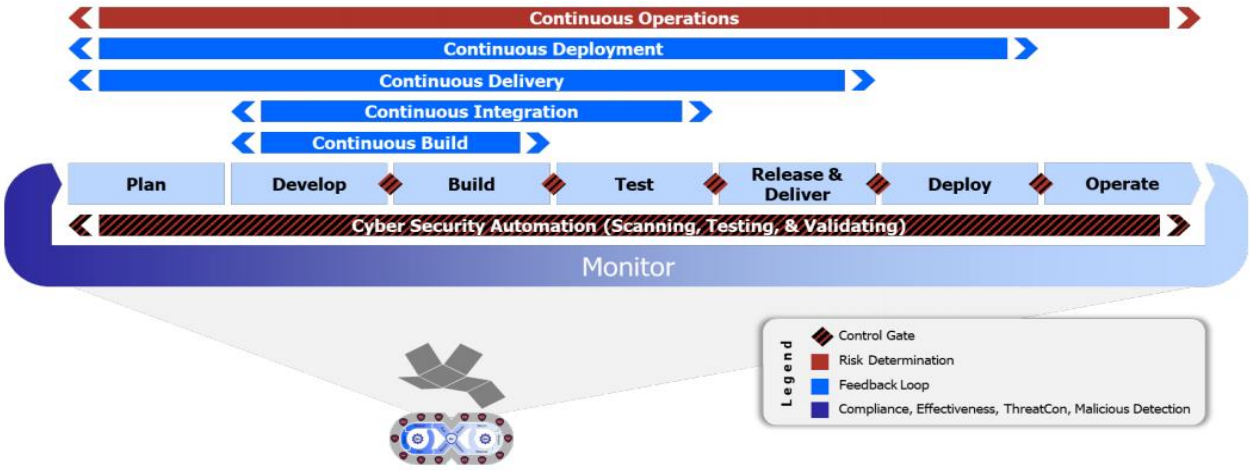


Figure 5 "Unfolded" DevSecOps Lifecycle Phases

Tal como muestra la figura 5, la metodología consiste en el anidamiento de varios ciclos de realimentación, los ciclos más grandes (líneas más largas) necesitan que los ciclos más pequeños se hayan ejecutado satisfactoriamente para poder completarse. En suma, se trata de automatizar todos los procesos que permitan la integración continua de las aportaciones de todos los programadores de la factoría, agilizando mucho la mejora de las aplicaciones. Simbólicamente: ir "esculpiendo" ininterrumpidamente, entre todos los programadores, el comportamiento deseado sobre las aplicaciones.

FEEDBACK LOOP	OBJETIVOS	DESCRIPCIÓN
Continuous Build	• **Compilación,** sin una compilación exitosa, los pasos adicionales son ilógicos e imposibles de completar.	• Revisión código: identificar riesgos de seguridad y código confuso. • Colisiones (merge): cuando varios programadores entregan simultáneamente la misma sección de código.
Continuous Integration	• **Fusión en rama principal**, integra el nuevo código sobre código base para lanzar pruebas de integración.	• Integraciones mínimas diarias: los desarrolladores presentan diariamente pequeños cambios. • Regresiones: detección temprana de errores de integración.
Continuous Delivery	• **Código listo para despliegue**.	• Aprobación y etiquetado de versión de código, aunque aún no se entregue a producción.
Continuous Deployment	• **Implementación**	• Despliegue en producción, primer punto de control fuera de la pipeline CI/CD.
Continuous Operation	• **Disponibilidad, rendimiento y riesgos operacionales**	• SLA (Service Level Agreement), disponibilidad 99'9% del tiempo. • Respuesta frente a caudal habitual de peticiones entrantes y márgenes de tolerancia.
Continuous Monitoring	• **Reconocer la totalidad del Sistema**, para valoración de impactos.	• Métricas agregadas, que aporten una visión de conjunto para valoración de impactos. • Eventos y amenazas, gestión de todos los eventos del sistema para evaluar tiempos y amenazas.

3.3.1.1 LA IMPORTANCIA DE LA ETAPA DE PLANIFICACIÓN

La etapa de planificación tiene como objetivo optimizar la gestión de tiempos, costes, riesgos e incidencias… y ha de adherirse a los principios "Lean Startup"[16]: fijar casos de uso principales sobre los que aplicar mejora continua incremental… tanto nuevas prestaciones como casos de uso secundarios.

> La etapa de planificación se repite antes de cada iteración del "Sprint".

Para optimizar la gestión de recursos, la etapa de planificación es la responsable de ir incorporando mejoras en la automatización de los procesos que tiene cada cadena de suministro DevSecOps (y sus puntos de control). De esta manera la factoría software va encapsulando, gradualmente, los procesos de entrega continua, hasta lograr que sean totalmente transparentes a los programadores, liberándoles de la burocracia necesaria para coordinar todos estos procesos y así puedan centrarse en la mejora de producto. Para ello son esenciales herramientas de comunicación que posibiliten recopilar las medidas a incorporar antes de cada nueva iteración del "Sprint"[17].

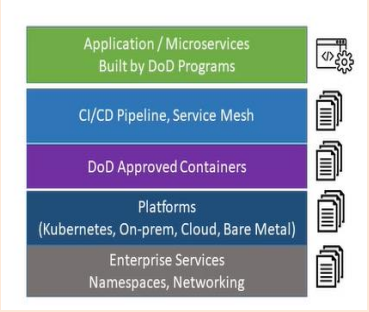
[16] *Eric Reis, "The Lean Startup,"* http://theleanstartup.com/principles

[17] *Ikujiro Nonaka y Takeuchi, Scrum Framework:* https://hbr.org/1986/01/the-new-new-product-development-game

3.3.2.- LA CIBERSEGURIDAD: MODELO ZERO-TRUST.

La abreviatura Sec en DevSecOps alude al hecho de incorporar dentro de la metodología de diseño de aplicaciones, estructuras y métodos de seguridad, en lugar de delegar la seguridad en los recipientes (los centros de datos) donde van a desplegarse. La identidad (ejm.: tarjeta SIM) y el reajuste automático de autorizaciones según contexto de conexión, sustituyen a las medidas de seguridad perimetral. Estrategia que surge como respuesta ante la ubicuidad y diversidad de las infraestructuras de despliegue modernas. Suele decirse que se aplica un modelo ZeroTrust de ciberseguridad en sustitución del antiguo modelo de Perímetro de Seguridad.

El modelo Zero-Trust se aplica durante las tres etapas del ciclo de vida de las aplicaciones de manera automatizada, para minimizar su impacto en el modo de trabajo diario de los programadores:

FASE	OBJETIVO	TECNOLOGÍAS
Diseño Servicio ***-Factoría-*** 	• **Limitaciones de acceso a los datos** para cada petición de API. • **Back-End inaccesibles**, solo a través de Front-End. • **Front-End con breve ciclo de vida**, para reducir el tiempo de exposición. 	• **Vulnerabilidades del código fuente** (Endor Labs *[Erl_23, Endor_22]*), diseño, gestión y monitorización de dependencias entre componentes. • **Sistema de permisos** del plano de datos (bases de datos, etc.). • **Contenedor OCI**, *[NSA_20]* condiciones de despliegue.
Despliegue Aplicación ***-Distro, Homologación-*** Tráfico East-West Microsegmentación 	• **Autenticación de Servicios:** fabricantes de servicios autorizados a desplegar. • **Autorización de Servicios:** políticas de lista blanca por servicio y cifrado de conexiones entrantes. 	• **Contenedor OCI**, condiciones de instanciación (*[NSA_20]* k8s, instanciación de PODs) • **Manifiesto** de la Malla de Servicios declarando servicios entrantes autorizados. • **Políticas de lista blanca** (Aporeto *[Palo-Alto_21]*), comunicaciones cifradas y lenguaje de programación de políticas de lista blanca por servicio. • **Malla de Servicios**, refrescos periódicos de los frontales siguiendo patrones de ganancia estadística.
Acceso al Ecosistema de Aplicaciones. ***-Cliente, Despliegue-*** cATO *[NIST_3]* Autorización Continua para Operar 	• **Layered cATO**: políticas RBAC (Role Based Access Controlled) y contexto de aplicación de las políticas, es decir, una tarjeta SIM por dispositivo y un sistema de escaneo continuo que detecta condiciones de acceso (el contexto de acceso) para recalibrar autorizaciones. • **Detección de patrones de comportamiento**, refrescar Front-End si se detecta algún comportamiento anómalo. 	• **eSIM, iSIM (embedded/integrated SIM)** *[ETSI_18]*: tarjetas identificativas que porta cada dispositivo. • **Software Define Network** que automatiza el despliegue de políticas de acceso a las aplicaciones para crear una estructura de contextos de autorización por capas. • **Nokia XDR**, detección de patrones de comportamiento y respuesta automática.

4.- PLATAFORMA DE ENTREGA CONTINUA: EL ENTORNO DE EJECUCIÓN DE SERVICIOS.

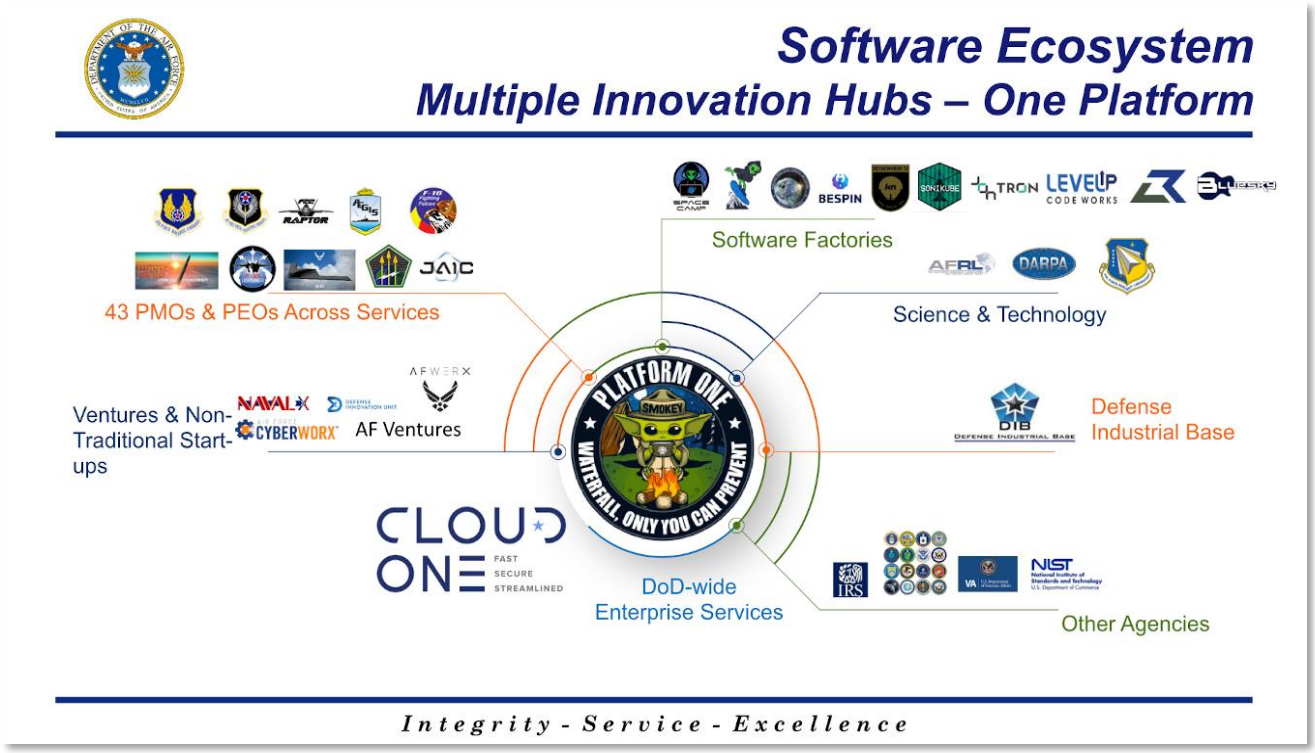
4.1.- EL ECOSISTEMA DE DESARROLLO SOFTWARE.

4.1.1.- EL DESPLIEGUE DE SERVICIOS: LA PLATAFORMA.

Tal como quiere representar la imagen, el uso de una misma plataforma, tanto por usuarios de aplicaciones como factorías software, permite establecer una metodología de trabajo común que garantiza unos estándares de calidad y seguridad en las aplicaciones resultantes.

Esta sección revisa el equipamiento necesario para poder adoptar ese sistema de recomendaciones de fabricación: una plataforma (Platform One[18] y Cloud One[19]) y un repositorio de contenedores (Repo One[20] y Iron Bank[21]), la base operativa de todo el tejido industrial que suministra aplicaciones al Departamento de Defensa de Estados Unidos *[DoD_1,2]*.

La sección comenzará revisando la anatomía de la plataforma de entrega continua (que ha de venir ensamblada de fábrica para que las factorías inicien su actividad productiva lo antes posible y al menor coste) y seguirá analizando el proceso de creación del repositorio de contenedores homologados que homogeneiza la calidad de las aplicaciones entregadas.



[18] ***Platform One, las Aplicaciones***, *el entorno de ejecución de Servicios:* https://p1.dso.mil

[19] ***Cloud One, los Centros de Datos***, *Infraestructura como Código:* https://www.cloud.mil/Cloud-One/

[20] ***Repo One,*** *el Repositorio de Fábrica:* https://repo1.dso.mil

[21] ***Iron Bank,*** *el Repositorio de Distribución:* https://ironbank.dso.mil/

4.1.2.- EL SUMINISTRO DE SERVICIOS: EL REPOSITORIO DE CONTENEDORES.

Una plataforma, y un repositorio de contenedores, la base de trabajo de las factorías de aplicaciones. Aún no existe un sistema oficial de distribución de contenedores (similar a los repositorios que ofrecen las distribuciones de Linux), solo hay proyectos como Quay de RedHat, que carece de procedimientos de homologación. Así que se tomará como referencia la metodología empleada en los repositorios Repo One y Iron Bank.

Las factorías de aplicaciones van liberando sus productos a un repositorio de imágenes de contenedor (Repo One). Sus cadenas de suministro software siguen toda una secuencia de pruebas automatizadas, con monitorización continua y puntos de control, de manera tal que los contenedores entregados vienen ya certificados de fábrica.

Una vez la factoría libera una versión de producto, ésta pasa por una cadena de procesos de homologación, a la que son sometidos todas las factorías software, se garantiza el despliegue sobre cualquier distribución de kubernetes y con las condiciones de instanciación establecidas para cumplir objetivos de seguridad. El resultado final, es el repositorio de distribución oficial (Iron Bank), donde sus contenedores están tanto certificados por fábrica como homologados y securizados para su distribución.

4.2.- LA PLATAFORMA: EL DESPLIEGUE DE SERVICIOS.

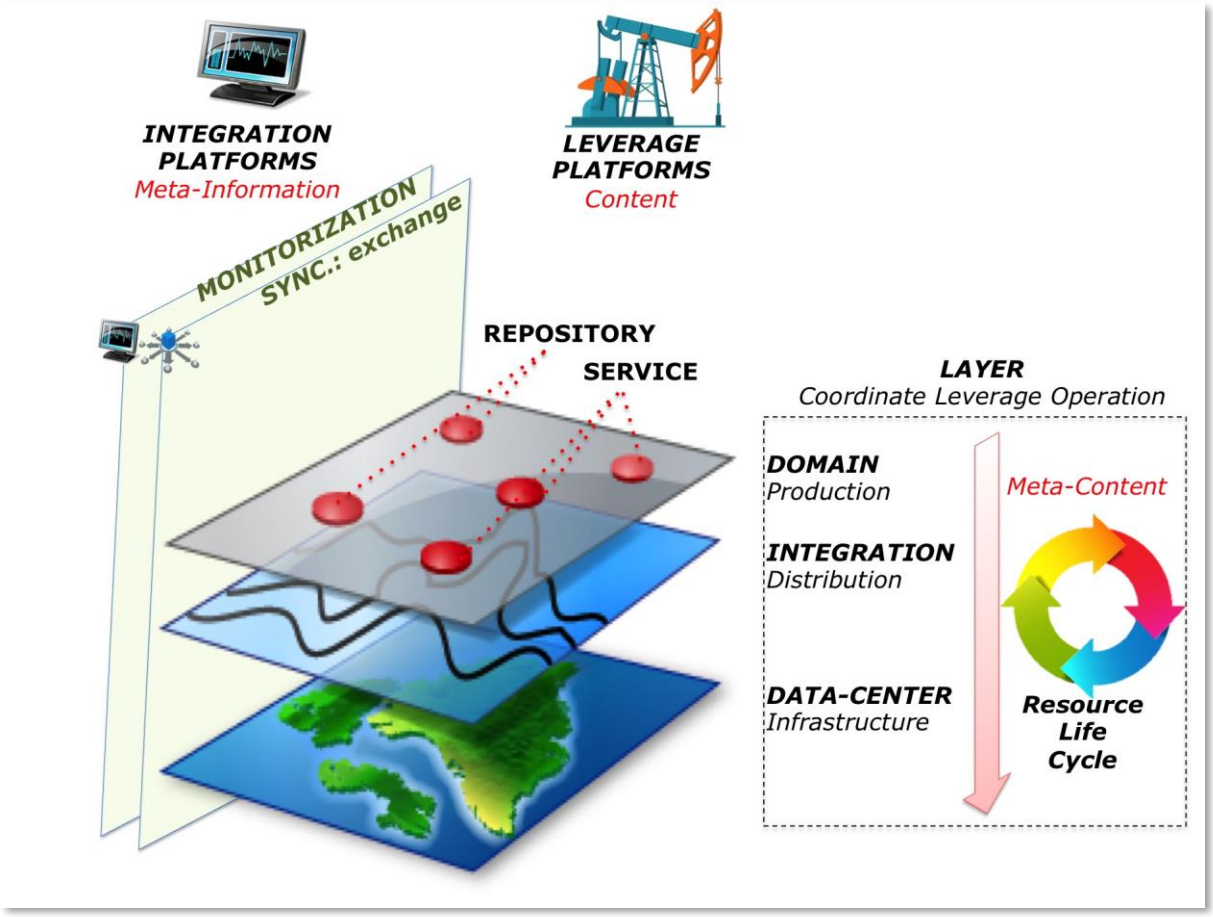
4.2.1.- MODELO DE SISTEMA: CAPAS DE OPERACIÓN Y ARTICULACIÓN.

La imagen representa la estructura de capas de cualquier plataforma. En horizontal las capas de explotación -u operación-, en vertical las capas de integración -o articulación-:

- **Capas de Operación**: distintos niveles de abstracción que permiten crear el valor resultante (en este caso, la aplicación orientada a servicios). Las capas quedan encapsuladas a través de interfaces, que han de mantenerse estables en el tiempo.

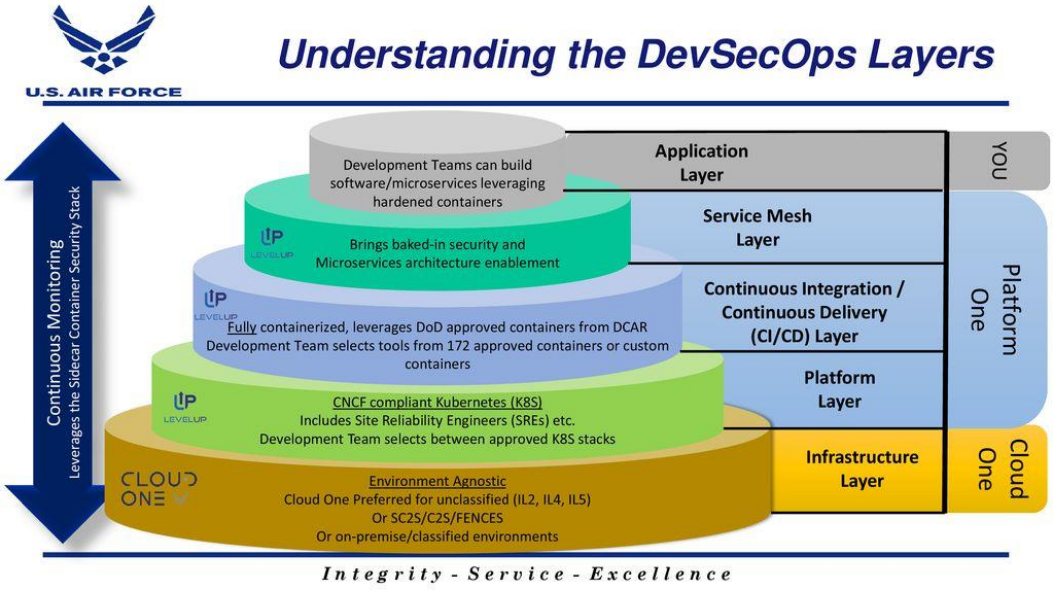
  > Es de una importancia crítica estandarizar las interfaces que encapsulan las distintas capas, para hacerlas totalmente independientes de las tecnologías de implementación (evitando un *vendor lock-in*, dependencia con fabricantes).

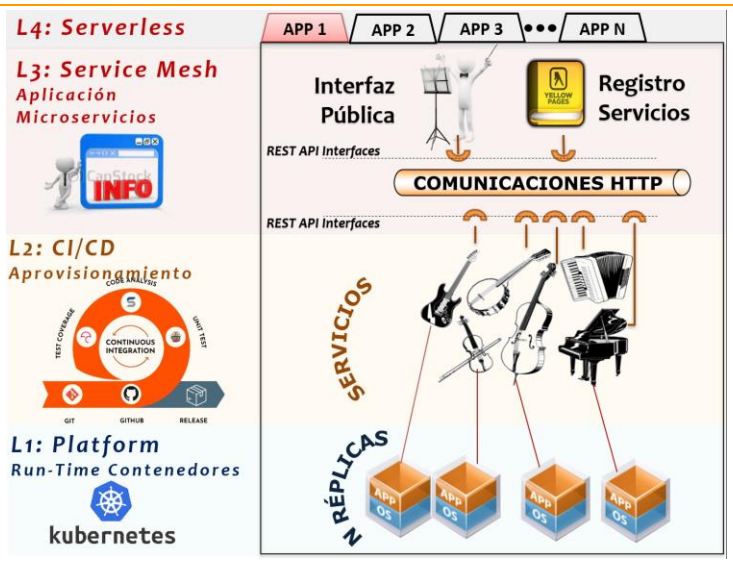
- **Capas de Articulación**: cada capa de operación tiene su plano de articulación que orquesta y coreografía la abstracción de la que se hace responsable esa capa. Las capas de articulación integran las articulaciones de todas las capas de operación para poder orquestar y coreografiar el ecosistema de aplicaciones orientadas a servicios de una manera centralizada y sencilla.



4.2.2.- CAPAS DE OPERACIÓN: DESPLIEGUE APLICACIONES.

La imagen representa la estructura de capas de explotación de las plataformas DevSecOps. El encapsulado de cada capa lo realiza una API (**A**pplication **P**rogramming **I**nterface). Esta propiedad comercialmente se identifica con la terminación *"as a Service"* (IaaS, PaaS, CaaS, etc.). Estos nombres comerciales son muy genéricos, por eso este documento empleará la nomenclatura de Platform One de las fuerzas aéreas de defensa de Estados Unidos, en cuyo documento de diseño se especifica qué funcionalidades encapsula cada capa, en otras palabras, qué prestaciones debemos esperar.





https://p1.dso.mil

https://www.cloud.mil/

CAPAS	OBJETIVO	TECNOLOGÍAS
L0 Infraestructura	• **Máquinas Físicas**: despliegue y control de una federación de racimos de ordenadores (o clústeres en inglés) desde una cabecera principal. El conjunto de prácticas L0 suelen llamarse *"Infraestructura como Código"*, aplicadas a través de una metodología GitOps.	• Cisco Application Centric Infrastructure (ACI) • Juniper Apstra • Arista CloudVision • Nokia Data-Center Fabric • OpenStack
L1 Plataforma	• **Terminaciones de Lógica**: instanciar las terminaciones de lógica (con sus contenedores) que componen un servicio sobre la federación de clústeres de la infraestructura L0.	• RedHat OpenShift • Novell Rancher • Canonical Charmed Kubernetes • VM Ware Tanzu
L2 CI/CD	• **Servicios**: aprovisionamiento y actualización continua de servicios desplegados en varias terminaciones de lógica (entre frontales y back-end) gestionadas por la plataforma L1.	• Tekton, fachada para cualquier sistema CI/CD • Jenkins, Jenkins X • GitLab
L3 Service Mesh	• **Aplicación**: automatización del despliegue de todos los servicios de una aplicación (en sus seis estrategias principales: recreate, ramped, blue/green, shadow, canary, a/b testing) y gestión de logs, en otras palabras, ensamblar los servicios aprovisionados por la capa L2.	• RedHat OpenShift Service Mesh • Istio • Traffik
L4 Serverless	• **Ecosistema Aplicaciones**: sistema de contextos para crear modelos de cohesión en el diseño de aplicaciones, es decir, facilitar la creación de ecosistemas de aplicaciones tal como hace un servidor de aplicaciones.	• RedHat OpenShift Serverless • Knative

4.2.3.- CAPAS DE ARTICULACIÓN: MONITORIZACIÓN Y CONTROL.

Las capas de articulación monitorizan y controlan de manera centralizada todo el ecosistema de aplicaciones orientadas a servicios. En la imagen de la página anterior se representan con una doble flecha azul, etiquetada con *"Continuous Monitoring"*, queriendo indicar que son transversales a todas las capas de operación, es decir, que coordinan las operaciones a lo largo de toda la estructura de capas y así se articula el ecosistema de aplicaciones de una manera sencilla.

CAPAS	OBJETIVO	TECNOLOGÍAS
A0 Coreografiado Ecosistema Aplicaciones	• **CCM – Plataforma de Monitorización Continua**: la monitorización de las aplicaciones depende del contenedor side-car que aporta la malla de servicios. A partir de ahí, se realiza una gestión centralizada de logs.	• Sidecar Container Security Stack • D2IQ
A1 Orquestación Ciclo de Vida de Aplicaciones	• **CCM – Plataforma de Control Centralizado**: el despliegue de aplicaciones se realiza a través de la gestión de contenedores side-car de la malla de servicios que automatiza el despliegue de todos los servicios que tiene cada aplicación.	• Sidecar Container Security Stack • D2IQ, consola para despliegue de aplicaciones.

4.3.- EL REPOSITORIO: LA ENTREGA CONTINUA.

4.3.1.- FACTORÍAS: LAS CADENAS DE SUMINISTRO SOFTWARE.

4.3.1.1 CICLO DE VIDA: ETAPAS E IMPERATIVOS.

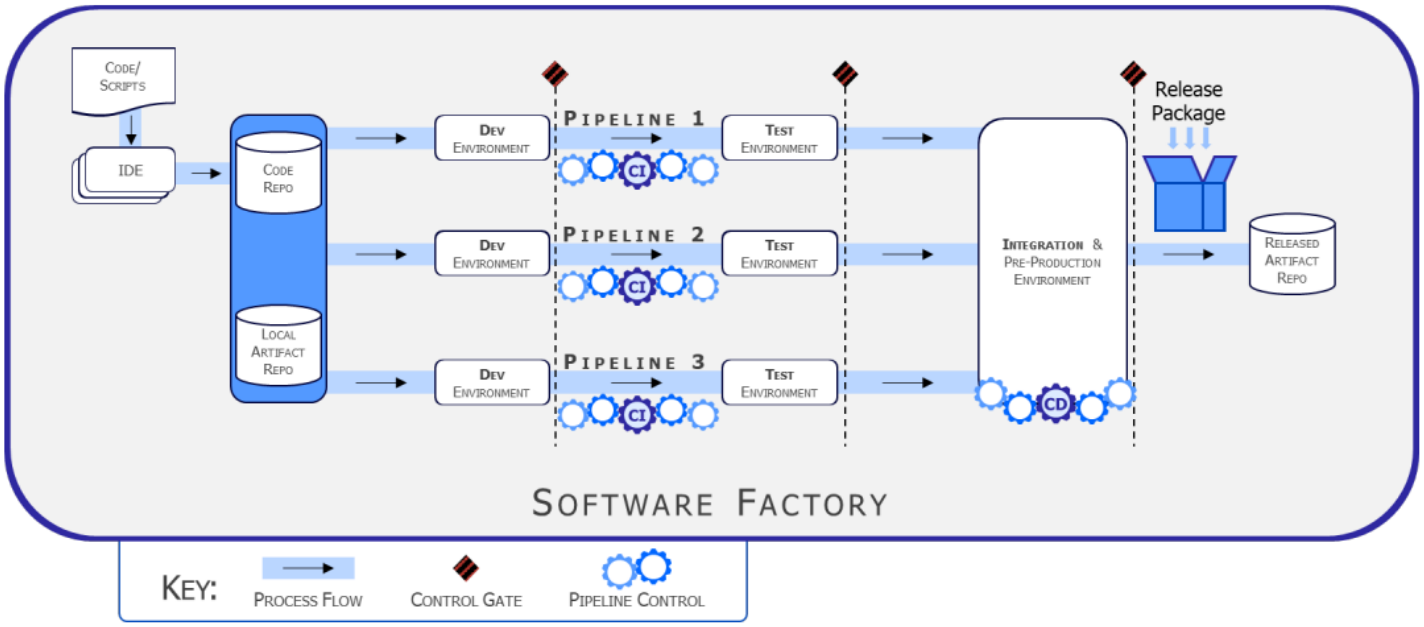


Figure 2 Normative Software Factory Construct

Las cadenas de suministro software han se adherirse a una serie de principios o imperativos:

ETAPA	OBJETIVOS	IMPERATIVOS
Desarrollo	• Agilizar desarrollo	• Control del árbol de dependencias entre componentes. • Actualizaciones frecuentes muy pequeñas. • Monitorización continua de cadenas de suministro y cambios. • Modernizar diseño de aplicaciones a arquitecturas orientadas a servicios. • Testeo continuo de la seguridad. • Servicios y componentes, han de reemplazarse completamente, no parcialmente, para evitar problemas de configuración.
Seguridad	• Zero-Trust	• Definir y aplicar tolerancias frente a riesgos, es decir, establecer superficie y tiempos de exposición de los datos. • Gestión centralizada de logs de aplicaciones.
Operación	• Despliegues rápidos.	• Marcha atrás en todo momento disponible. • Monitorización continua de plataforma. • Despliegues Blue/Green para versiones de aplicación. • Despliegues Canary para funcionalidades incrementales.

4.3.1.2 DIAGRAMA DE FLUJO: DINÁMICAS Y HERRAMIENTAS

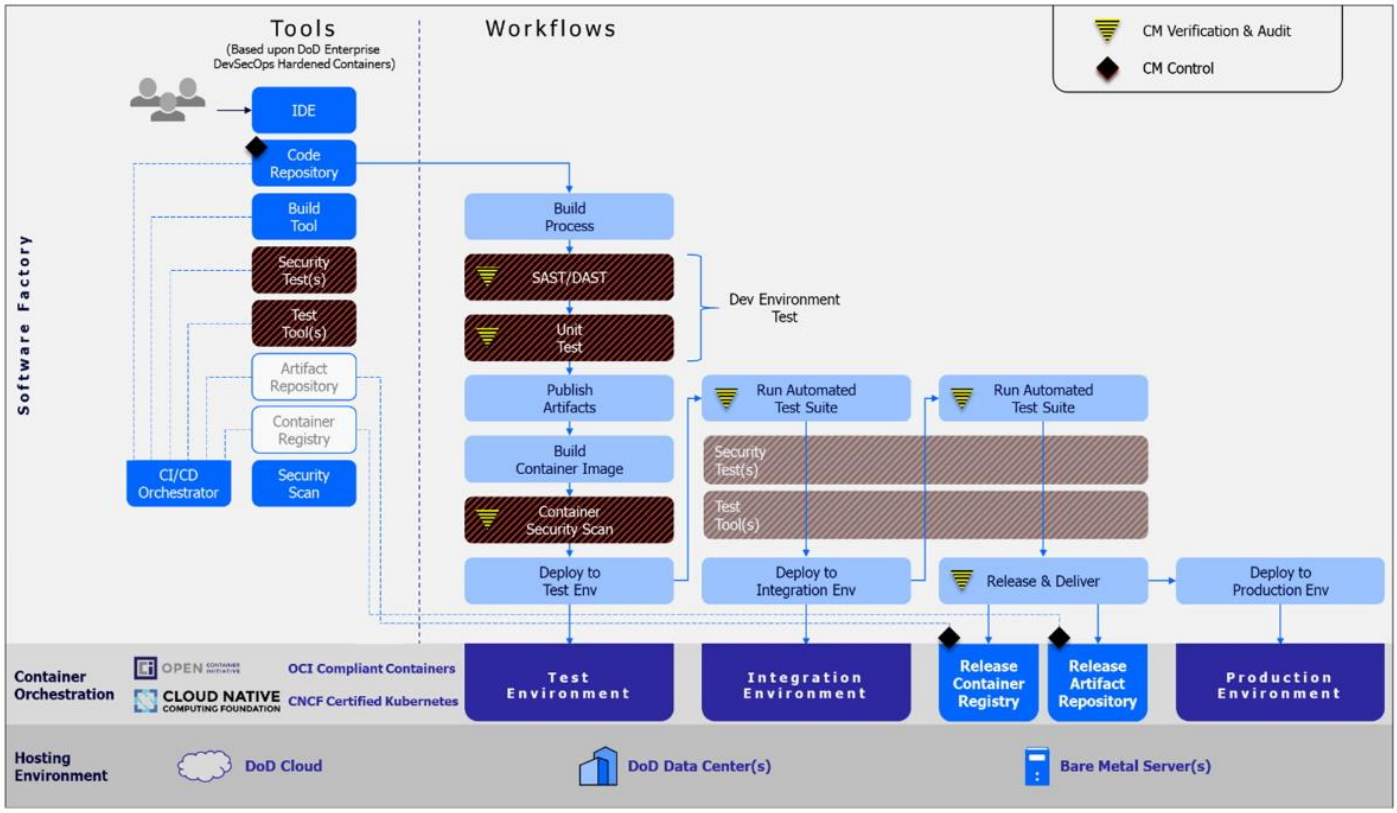


Figure 5: Containerized Software Factory Reference Design

4.3.1.3 CIBERSEGURIDAD: PUNTOS DE CONTROL EN LAS CADENAS DE SUMINISTRO SOFTWARE.

Cada factoría desarrolla sus propias pruebas de ciberseguridad ajustadas al producto que elabora. Sin embargo, las cadenas de suministro han de establecer puntos de control de calidad, o puertas de control, tras cada etapa de la cadena de suministro (representados por un rombo en la imagen), donde la factoría verifica el cumplimiento de los objetivos de seguridad previamente establecidos. La imagen representa algunas familias de pruebas que suelen aplicarse en esas puertas de control. Inicialmente se aplican manualmente, pero gradualmente van automatizándose hasta agilizar la entrega y facilitar evaluación de riesgos e impactos:

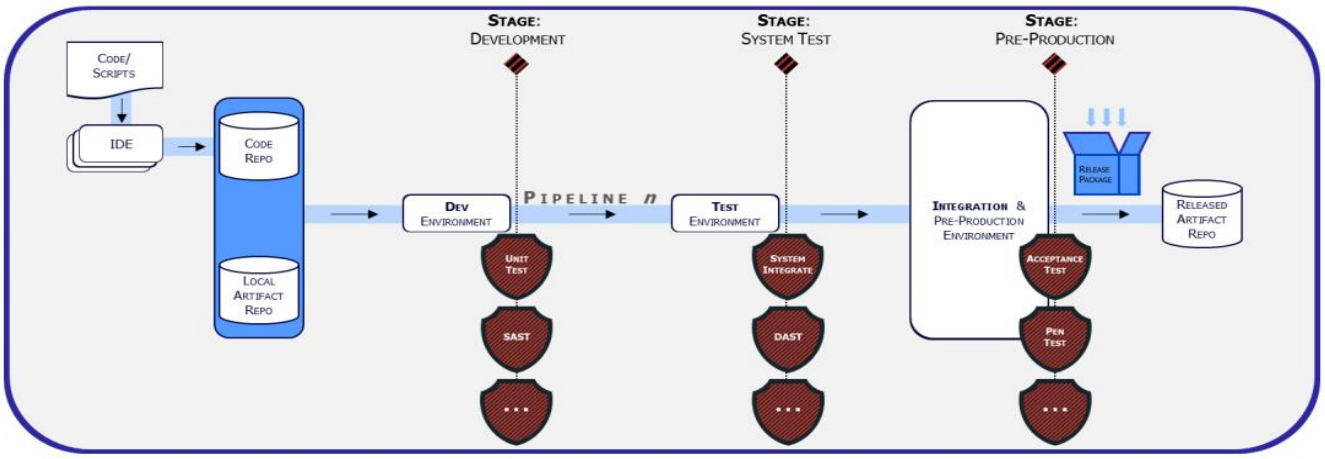


Figure 6 Notional expansion of a single DevSecOps software factory pipeline

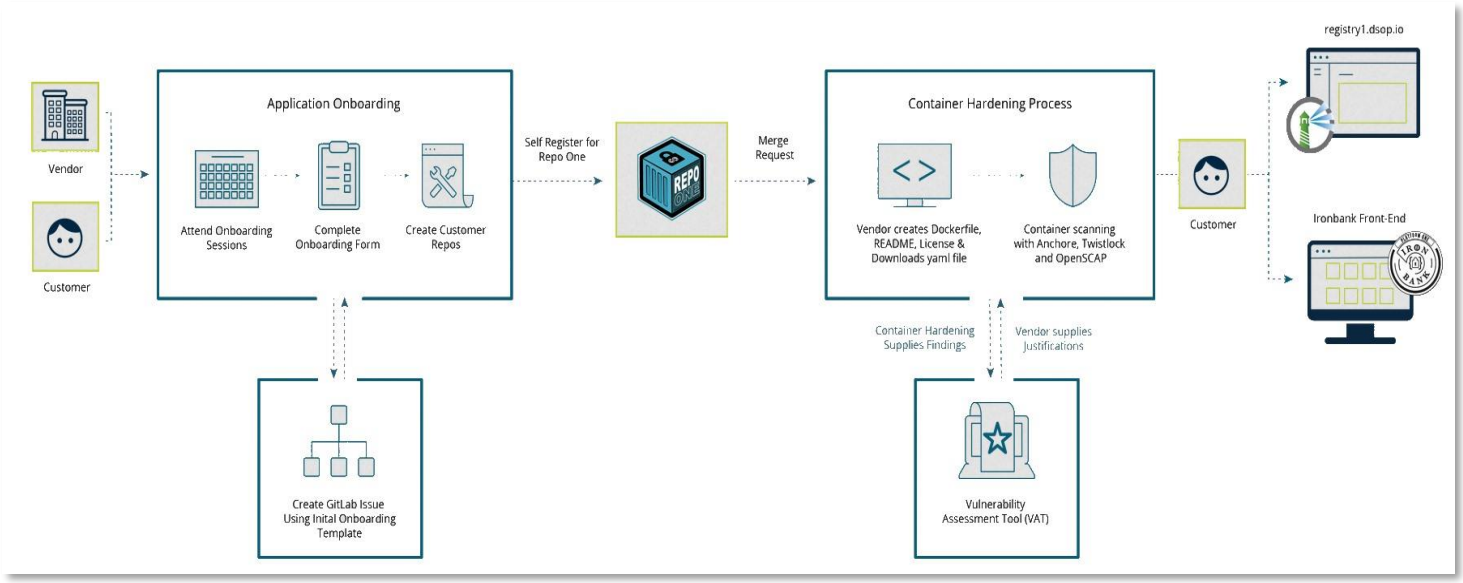
ETAPA	OBJETIVOS	TÉCNICA
Desarrollo	• **Componentes**, Seguridad y Versatilidad	• Pruebas Unitarias • Static Application Security Testing (SAST)
Pruebas de Sistema	• **Sistema Final**, Seguridad de Funcionalidades.	• Pruebas Integración • Dynamic Application Security Testing (DAST)
Pruebas de Certificación	• **Certificación en entorno Real**, recomendaciones de despliegue.	• Pruebas de Aceptación • Pruebas de Certificación

El modelo Zero-Trust implica agregar pruebas específicas para la seguridad: pruebas SAST y DAST.

PRUEBA	ETAPA ZERO-TRUST	OBJETIVOS
SAST=Static Application Security Testing	• **Diseño Servicio**	• Servicio: Superficie de exposición de los datos de cada servicio… restricciones de acceso a los datos desde los frontales.
DAST=Dynamic Application Security Testing	• **Despliegue Aplicación**	• Aplicación: Superficie y tiempo de exposición de los datos ○ *Microsegmentación*, visibilidad entre servicios ○ *Diseño del refresco de los frontales*, minimizar la superficie de exposición con esquemas de alternancia.
Pruebas Certificación	• **Layered cATO**	• Ecosistema Aplicaciones: usuarios y contextos de conexión… verificar la estructura de capas de autorización.

4.3.2.- DISTRIBUCIÓN: LA HOMOLOGACIÓN DE CONTENEDORES.

La imagen representa el ciclo de homologación de contenedores a las que son sometidas las factorías software asociadas al Departamento de Defensa de Estados Unidos. Esta homologación es equiparable al trabajo que realizan las distros Linux para presentar su sistema de repositorios[22]. Las factorías emplearán Iron Bank como sistema de distribución de contenedores homologados (similar a los repositorios Linux, pero para contenedores), simplificando la gestión y despliegue de sus aplicaciones, lo que redunda en una mayor agilidad y calidad en sus desarrollos.



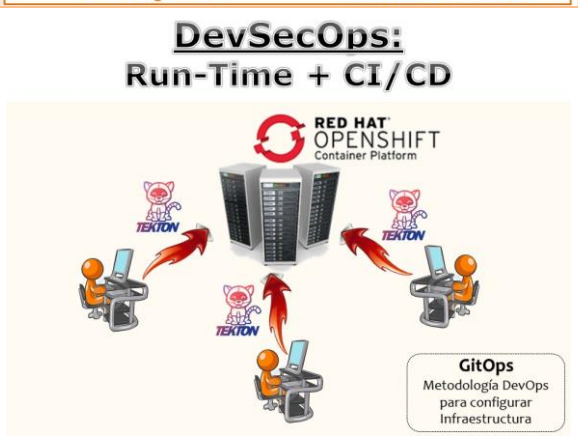
[22] *Debian Handbook, "el ciclo de vida de liberación de nuevas versiones":* https://debian-handbook.info/browse/stable/sect.release-lifecycle.html

5.- LOS ENTORNOS DE TRABAJO: CONSUMIDORES Y FACTORÍAS DE APLICACIONES.

5.1.- CASOS DE USOS Y METODOLOGÍA.

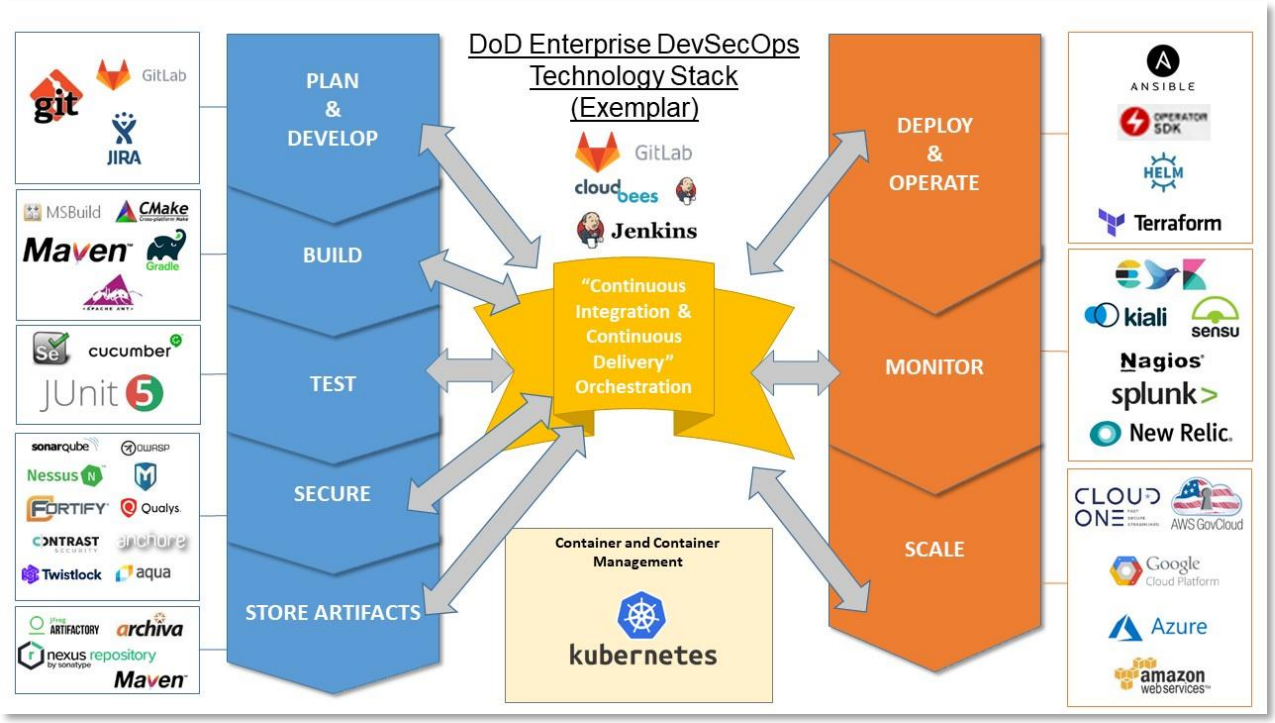
5.1.1.- CASOS DE USO: CONSUMIDORES Y FACTORÍAS DE APLICACIONES

El repositorio de contenedores es gestionado por cada fabricante, mientras que los consumidores despliegan sus aplicaciones a partir de ellos. Así que aparecen dos casos de usos para la misma metodología de entrega continua, los consumidores que despliegan aplicaciones y los fabricantes que crean y mantienen esos repositorios de servicios.

CASO USO	OBJETIVO	DESCRIPCIÓN
Consumidores de Aplicaciones Despliegue de aplicaciones en Cadenas de Sucursales.	• **Cabecera:** Pruebas de Integración para verificar los despliegues de aplicación. • **Cadena de Centros de Datos[23]:** propagación de actualizaciones desde la cabecera de pruebas a toda la cadena de sucursales.	
Factorías Software Suministro de Servicios.	• **Maquetas I+D:** donde los desarrolladores pruebas partes del sistema, con total libertad. Cada grupo de desarrolladores puede tener su propia maqueta. • **Maquetas de Pruebas:** existen distintos tipos de maquetas, según el tipo de pruebas que van a ejecutarse: ○ Integración: para verificar sistema completo. ○ Carga: pruebas de estrés sobre el sistema completo. ○ Operación: emular entornos de cliente final.	

[23] ***Lt. Gen. Jack Shanahan*** *(director del centro de inteligencia artificial del Departamento de Defensa de Estados Unidos),* ***"la falta de nube empresarial"*** https://fcw.com/it-modernization/2020/05/pentagons-ai-chief-lack-of-enterprise-cloud-has-slowed-us-down/196057/

5.1.2.- METODOLOGÍA: MAPA DE HERRAMIENTAS DEVSECOPS.



5.1.3.- MÉTRICAS DEV-OPS: DORA-GOOGLE

DevOps proporciona mejoras tangibles y demostrables frente a las metodologías tradicionales de desarrollo software. La efectividad puede ser evaluada con métricas de tiempo y estabilidad, ambas establecidas de manera muy precisa en el DORA (DevOps Research and Assessment Quick Check)[24], que se complementan con las "*Four Golden Signals*" establecidas por Google[25]. Primero se revisarán las métricas, después los métodos que permiten su medición.

Las métricas DORA miden la eficacia de los procesos de entrega que realizan las factorías software, mientras que las métricas Google miden el rendimiento de las aplicaciones desplegadas en las diferentes plataformas. Dos perspectivas que han de combinarse para poder tener una evaluación completa, pero que se recaban mediante procedimientos completamente diferentes.

METRICAS DORA – Dev	MÉTRICAS GOOGLE – Ops
• **Frecuencia de Despliegues**:	• **Latencia**:
• **Lead Time**:	• **Tráfico**:
• **MTTR – Tiempo Medio Resolución Errores**:	• **Errores**:
• **CFR – Tasas de Errores durante Actualizaciones**:	• **Saturación**:

[24] *DevOps Research and Assessment Quick Check:* https://www.devopsresearch.com

[25] *Google, The Four Golden Signals:* https://sre.google/sre-book/monitoring-distributed-systems/

5.2.- FACTORÍA: DESPLIEGUE PROCESOS DEVSECOPS.

5.2.1.- MAPA DE HERRAMIENTAS MÍNIMAS EN FACTORÍAS

- **Sistema de Colaboración en Equipo**: como Atlassian Confluence, una wiki colaborativa que permite a los equipos compartir documentación de manera que se integra totalmente con el sistema de gestión de incidenticas. La herramienta elegida deberá soportar "multi-tenancy" (aislamiento entre proyectos).
- **Sistema de Seguimiento de Objetivos**: como Atlassian Jira, un sistema robusto de seguimiento de objetivos que ha de ser ajustable a las necesidades de cada equipo de desarrolladores. La herramienta elegida deberá soportar "multi-tenancy".
- **Entorno Integrado de Desarrollo (IDE)**: hay muchos IDEs, como por ejemplo Visual Studio Code ajustado para construir y depurar aplicaciones nativas de nube.
- **Repositorio de Código Fuente**: es indispensable que el control de versiones en el que se basan estos repositorio sea solución distribuida (como git) para aportar la agilidad necesaria en las factorías software que los modelos centralizados de gestión de versiones no pueden ofrecer. Ejemplos comunes, GitLab o Atlassian Bitbucket.
- **Herramientas de Construcción**: la herramientas de construcción y compilado son una componente esencial de las tuberías CI/CD. Ejemplos, seria Jenkins.
- **Repositorio de Artefactos**: debe integrarse en las herramientas de construcción, ofreciendo acceso tanto a las dependencias que cada artefacto necesita para completa su construcción, como almacenamiento de los artefactos que salen de la herramienta de construcción.
- **Herramientas y Suites de Automatización de Pruebas**: las características de la herramienta de pruebas debe ajustarse al tipo de aplicaciones que van a construirse.
- **Herramientas de Reportes de Cobertura de Pruebas**: se anima a los equipos a mantener un umbral alto de cobertura de código con pruebas asociadas, y es vital reportar qué porcentaje de pruebas se ha superado en cada momento.
- **Pruebas de Seguridad Dinámicas y Estáticas (SAST y DAST)**: herramientas avanzadas pueden detectar un amplio rango de problemas, desde algoritmos pobremente elaborados hasta agujeros de seguridad.
- **Estrategias de Logging**: herramientas agregación, análisis y auditoría de logs es obligatorio incorporarlas.
- **Monitorización Continua**: InfoSec y monitorización operacional de toda la factoría software es algo necesario, y deben especificarse tanto herramientas como mecanismos que cada factoría emplea.
- **Estrategias de Alertas y Notificación**: las tuberías CI/CD han de tener acceso a capacidades de alertas y notificaciones que puedan ser activadas desde las herramientas de gestión de incidencias, herramientas de construcción, herramientas de testeo automático, SAST/DAST y herramientas de monitorización.

5.2.2.- MEDICIONES DORA: LOS SEIS INDICADORES

- **La Mejora Continua**: La industria continua mejorando, especialmente entre los mejores actores. Hay ciertas capacidades clave para lograr la excelencia.
- **La evolución tecnológica depende del desarrollo de aplicaciones**: La entrega de software fiable, seguro y de forma ágil es el corazón de la tecnología.
- **Creación de comunidades estables**: Las mejores estrategias para escalar prácticas DevOps en las organizaciones se concentran en soluciones estructurales que crean comunidad, en niveles altos y bajos de la organización, incluyendo prácticas comunitarias y pruebas de concepto.
- **Nube como elemento diferenciador**: las factorías software que emplean computación en la nube tienen claras ventajas.
- **La productividad mejora la calidad de vida del trabajador**: un trabajo bien planificado y repartido, con estrategias a largo plazo, mejora la eficiencia de cada individuo.
- **Políticas de cambios adecuadas**: favorecer cambios pequeños, una evolución progresiva, implica un resultado estable.

5.2.3.- ENTORNOS DE TRABAJO.

5.2.3.1 REPOSITORIO LOCAL

5.2.3.2 PLATAFORMAS: MAQUETAS DE DESARROLLO Y PRUEBAS

5.3.- CONSUMIDOR: LA ENTREGA CONTINUA.

5.3.1.- MEDICIONES GOOGLE.

5.3.2.- ENTORNOS DE TRABAJO.

5.3.2.1 REPOSITORIO LOCAL

5.3.2.2 PLATAFORMAS: CABECERA Y SUCURSALES

6.- ESTADO DEL ARTE.

Debido al alto coste de estas plataformas de nube, los fabricantes de aplicaciones tienen que suscribirse a agentes externos para poder desarrollar su actividad, en dos modalidades básicas:

- **Plataforma externa**: alquilar recursos a aquellos operadores de centros de datos con grandes capacidades de computación en paralelo (como Google o Amazon).

- **Plataforma propia**: emplear servicios externos de gestión y automatización de su propia maquinaria (como D2IQ), o conformarse con plataformas de reducidísimas capacidades a un coste de mantenimiento desorbitado.

Sirva esta introducción como recopilación muy resumida de las mejores prácticas del sector, en la esperanza de que, gracias a comprender el engranaje de piezas, se facilite el profundizar en cada una de ellas.

7.- BIBLIOGRAFÍA

[1]	Platform One, referencias de diseño, *US Defense*	https://software.af.mil/dsop/documents/ https://p1.dso.mil/ https://cloudone.af.mil/
[2]	Iron Bank, repositorio de contenedores, *US Defense*	https://p1.dso.mil/products/iron-bank https://repo1.dso.mil/dsop
[3]	cATO, Continous Authorization to Operate, *NIST*	https://media.dau.edu/media/t/1_r89h14za https://csrc.nist.gov/glossary/term/authorization_to_operate
[4]	Services Oriented Architecture, *Thomas Erl*	https://es.arcitura.com/book/service-oriented-architecture-concepts-technology-and-design/chapter-descriptions/
[5]	Patterns of Enterprise Application Architecture, *Martin Fowler*	https://www.martinfowler.com/books/eaa.html
[6]	Enterprise Integration Patterns, *Gregor Hophe*	https://www.enterpriseintegrationpatterns.com/
[7]	The Distributed Systems, Apstra Network Operating System, *David Cheriton*	https://web.archive.org/web/20160304074443/http://gregorio.stanford.edu/
[8]	Dependency Injection Principles, Practices, and Patterns, *Steven van Deursen*	https://www.oreilly.com/library/view/dependency-injection-principles/9781617294730/
[9]	Patrones Despliegue Aplicaciones en Nube, *Thomas Erl*	https://patterns.arcitura.com/cloud-computing-patterns
[10]	Patrones SOA, *Thomas Erl*	https://patterns.arcitura.com/soa-patterns
[11]	Oracle Certified Professional, Java EE 7 Application Developer, *Oracle University*	https://education.oracle.com/en/oracle-certified-professional-java-ee-7-application-developer/trackp_900
[12]	SOA with Java, *Thomas Erl*	https://es.arcitura.com/book/soa-with-java-realizing-service-orientation-with-java-technologies/order/
[13]	J2EE, Standards for Java Platform Enterprise Edition, *Eclipse Foundation*	https://jakarta.ee/
[14]	Microprofile, Standards para microservicios en Java, *Apache Foundation*	https://microprofile.io/ https://quarkus.io/
[15]	Knative, Serverless Approach, *Cloud Native Foundation*	https://knative.dev/docs/
[16]	Defensa Acquisition University, *US Defense*	https://software.af.mil/training/ https://www.dau.edu/ https://media.dau.edu/playlist/details/1_iu6ulm7r
[17]	Certificación Arquitecto SOA, *Arcitura IT*	https://www.arcitura.com/soa-school/certifications/certified-soa-architect/
[18]	iSIM, eSIM standards, *ETSI*	https://www.etsi.org/technologies/sim
[19]	Design Microservices the Right Way, *Michael Bryzek*	https://youtu.be/j6ow-UemzBc
[20]	Kubernetes Hardening Guidance, *US National Security Agency*	https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/2716980/nsa-cisa-release-kubernetes-hardening-guidance
[21]	Aporeto, Identity Based Microsegmentation, *Palo Alto Networks*	https://www.paloaltonetworks.com/blog/prisma-cloud/aporeto-integration-prisma-cloud/
[22]	Dependecy Management System, *Endor Labs*	https://www.endorlabs.com/
[23]	Web Service Contract, Design and Versioning, *Thomas Erl*	https://es.arcitura.com/book/web-service-contract-design-and-versioning-for-soa/
[24]	Resilient SOA, *Fraunhofer Institute*	https://www.iks.fraunhofer.de/en/research/resilient-soa.html