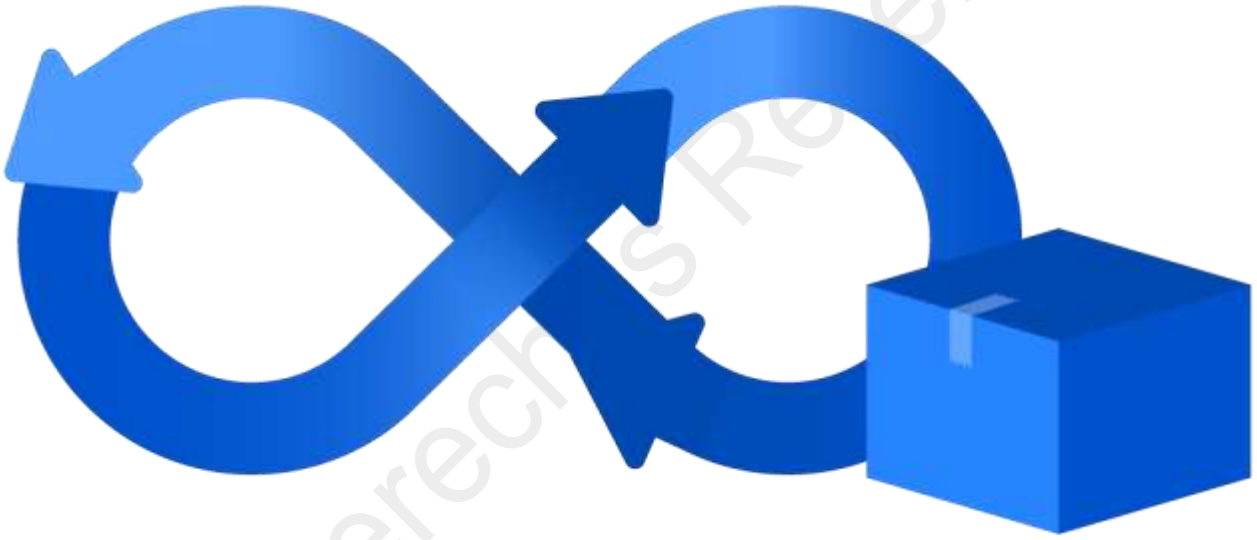
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

DevOps, Integridad y Agilidad Continúa

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Unidad 4: Entrega Continua



Presentación:

En esta unidad se identificarán las bases en las cuales se sostiene la práctica de la Entrega Continua (Continuous Delivery), observaremos sus principales aspectos relacionados a las herramientas y a la organización.

Resaltaremos la importancia de tener una infraestructura versionada para incorporar su gestión dentro del pipeline de desarrollo de manera automatizada y posibilitar lo que se conoce como arquitectura inmutable.

Objetivos:

Que los participantes:

- Conozcan los principios de la práctica de Entrega Continua.
- Conozcan los aspectos principales de la infraestructura versionada y sus beneficios.
- Identifiquen la importancia de la gestión de la configuración.
- Identifiquen estrategias que conciernen a los despliegues en producción.
- Identifiquen los conceptos subyacentes de un pipeline development.

Bloques temáticos:

1. Introducción a Continuous Delivery

2. Infraestructura versionada

3. Estrategias

4. Práctica

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

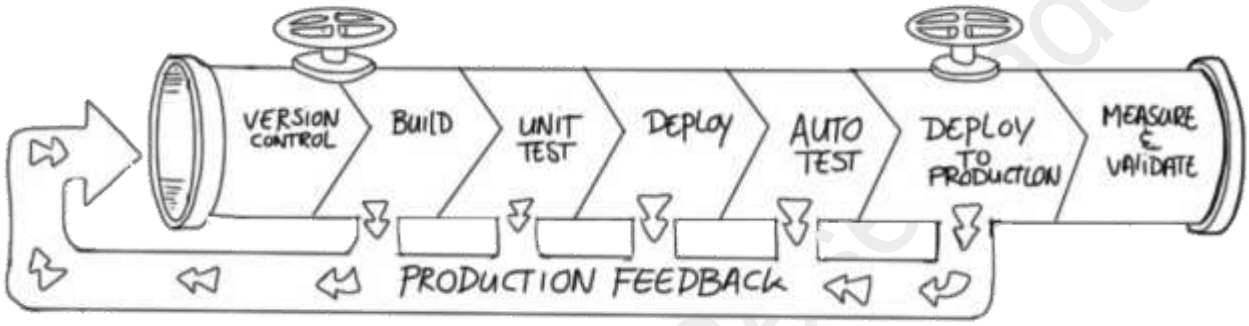
El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. Introducción a Continuous Delivery



La entrega continua es más que la compra de una herramienta de software que permita hacer entregas a nuestros clientes de una manera rápida. La entrega continua o Continuous Delivery es un paradigma distinto en el cual se sostiene el negocio de una organización digitalizada o que posee una necesidad de ello.

El principal obstáculo que trae aparejado la Entrega Continua en una organización que no tiene el suficiente nivel de madurez para llevar adelante este paradigma es la falta de lineamientos dentro los niveles de management y los equipos (silos). Los equipos de desarrollo están presionados para entregar software de una manera rápida, performante, con calidad y en algunos casos siguiendo ciertas normas regulatorias. Por otro lado, están los equipos de implementaciones (operaciones / infraestructura) que analizan de cada implementación en producción su riesgo y en el mejor de los casos obtienen ciertas métricas de experiencias anteriores, datos muy importantes para una práctica concisa de DevOps ya que sin la medición y el monitoreo no se sabría la situación actual de la aplicación o si los features que se entregaron realmente están generando valor (recordar de unidades anteriores la visión Lean y su ciclo).

En organizaciones con poca madurez en los aspectos de DevOps, existe una tensión entre los equipos de desarrollo y operaciones provocada por la falta de lineamientos

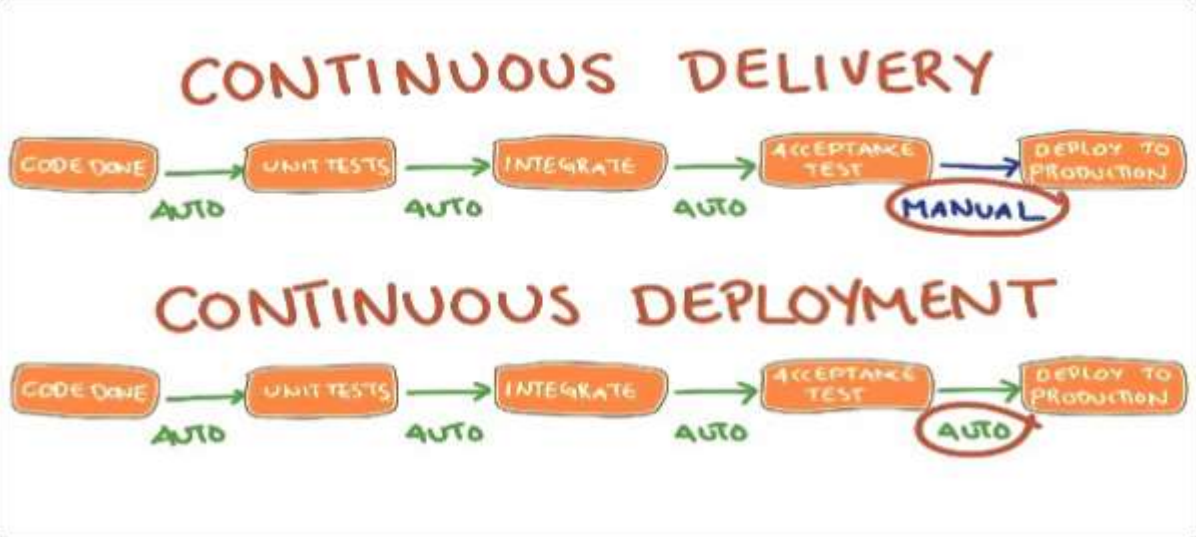
estratégicos para obtener una ganancia de la manera más rápida posible y aprender de lo recientemente lanzado a producción (medirlo) para saber si hay que adaptar el producto (aprendizaje), y a su vez tener los ambientes de producción estables. Para ello, el pipeline de desarrollo se diseña de principio a fin involucrando a todas las partes de la organización que van a agregar valor. Durante el diseño se procura realizar un lineamiento de objetivos con metas de negocio que sea transversal a las áreas (Negocio y Tecnología). El abordaje, a través de un pipeline permite comenzar a diluir las barreras que existen en los silos de organizaciones medianas a grandes, mediante la colaboración de equipos multidisciplinarios que realizan diferentes actividades.

La principal ventaja de las prácticas y técnicas vistas hasta aquí son utilizadas para diseñar un pipeline de desarrollo que permita disminuir el riesgo de las implementaciones, aumentar el feedback de una manera rápida y no tan costosa a través del desarrollo iterativo e incremental, con menores defectos y la automatización de pruebas e implementaciones reduciendo el ciclo de entrega.

La identificación de riesgos es un aspecto importante para establecer Entrega Continua y sobre todo para la organización que la adopte. Los principales aspectos que se deben tener en cuenta, no siendo los únicos, son los siguientes:

- Identificación de los riesgos principales del proyecto/producto
- Estrategias para la mitigación de los riesgos identificados
- Estrategia para el seguimiento de los riesgos durante el curso del proyecto/producto
- Definir una estructura para el reporte de estado de los equipos

Estos aspectos, en general, se suelen tratar en etapas iniciales del proyecto bastante antes del comienzo del desarrollo en la cual se conversan entre todos los involucrados de la organización, tanto el negocio como tecnología (comerciales, miembros de los equipos de desarrollo, de operaciones, de seguridad, de testing, ejecutivos, gerentes, etc.) y se deben continuar a lo largo de todo el producto, ya que los documentos que generen al inicio son documentos vivos, los cuales seguramente vayan cambiando a lo largo de la vida del producto y es suma importancia que se actualicen.
En las etapas iniciales de un producto se intenta tomar la menor cantidad de decisiones que pueden afrontar grandes riesgos, ya que se da por sentado (y transparentado) que las decisiones tomadas se basan en asunciones y no en hechos.



La esencia de la Entrega Continua es que la organización pueda tener la capacidad de entregar pequeños cambios de una manera incremental y continua, de manera de no sólo entregar software rápidamente, sino que además reduzca los riesgos asociados al introducir cambios en el sistema. La manera en que lo realizará será con el llamado “One Click Deploy” (implementar con un sólo click). Esto significa que se automatiza todo el pipeline de desarrollo incluyendo la salida a producción, pero con la diferencia de que alguien de la organización tendrá la posibilidad de elegir en qué momento implementar el software que está listo para salir a producción, es decir que para que se realice el pasaje a producción se requiere una aprobación e intervención manual.

La diferencia entre implementaciones continuas (Continuous Deployment) y Entregas Continuas es que en el caso de las primeras el ciclo está automatizado y es directo hasta la salida a producción. Es decir que no es necesaria la intervención de una persona para la salida a producción. Este tipo de práctica requiere un nivel de madurez muy alto en la organización asociado a la cultura ágil y de DevOps.

2. Infraestructura versionada

Se dice infraestructura versionada al proceso de tener los ambientes de desarrollo, pruebas, configuraciones, virtualización de hardware, redes y manejo de datos gestionados a través de un sistema de control de versiones que permita aplicar cambios a través de un proceso automatizado.

Este es el norte en una organización basada en DevOps, sin embargo, llegar hasta allí requiere de mucho trabajo y esfuerzo, por lo tanto, aunque la organización no posea un aprovisionamiento de infraestructura embebido en un sistema de control de versiones se debería preguntar: cómo formalizar un proceso una vez que se realizó el aprovisionamiento de infraestructura. Y sí se está en un ciclo de desarrollo e implementaciones con un producto en producción, de qué manera se realizarán cambios en la infraestructura una vez que la aplicación está en uso por los clientes. Cómo se realizarán las actualizaciones de software.

Además, de contestar algunos interrogantes como los anteriores hay ciertos aspectos que se deben tener en cuenta a la hora de implementar la gestión de la infraestructura mediante un sistema de control de versiones. Algunos de esos aspectos son:

- Definiciones de instalación de sistemas operativos
- Versiones de aprovisionamiento de software
- Configuraciones del software que se aprovisiona
- Configuraciones generales tales como DNS, routers, firewalls, SMTP, etc.
- Cualquier script o programa desarrollado para gestionar la infraestructura

Todos estos aspectos son candidatos para estar en un sistema de control de versiones y embeberlos dentro de la automatización del pipeline. En el caso del ámbito que le concierne al pipeline, la infraestructura debe inspeccionar principalmente tres aspectos.

El primero, es que antes de que una aplicación sea implementada en un ambiente se debe probar que las configuraciones del ambiente (software, configuraciones generales, aprovisionamiento, pruebas funcionales y no funcionales etc.) sean las definidas en el sistema de control de versiones.
Segundo, estas definiciones deben ser aplicadas a los distintos ambientes (desarrollo, pruebas y producción) de una manera automática, recreando la infraestructura a partir de dichas definiciones que surgen del versionado y la herramienta de gestión.
Y, por último, el pipeline en el momento de hacer la implementación debe ejecutar ciertas pruebas para verificar que la infraestructura se implementó de una manera correcta.

Una de las complicaciones que se pueden presentar al tener la gestión de la infraestructura mediante un control de versiones y automatizada, es cuando hay dependencias entre las configuraciones de distintas aplicaciones que son compartidas en un mismo ambiente, ya sea de pruebas o productivo. Con lo cual, si bien es un buen enfoque tener la infraestructura versionada a través del pipeline de desarrollo de un producto, la gestión de dependencias de la infraestructura requiere de un mayor esfuerzo de coordinación. Una posible solución es crear un pipeline separado y compartido para la infraestructura de manera de poder gestionar las dependencias entre la infraestructura de las distintas que poseen las aplicaciones y embeberlo en el sistema de control de versiones. Es decir que la implementación de este pipeline afectará múltiples aplicaciones, por lo que hay que tener especial cuidado en la manera en que se gestiona.

La ventaja que trae la infraestructura versionada es la posibilidad de implementar y

rehacer nuestra infraestructura desde cero en cualquier momento y dejarla en un estado "saludable" desde el inicio a través de un proceso automatizado. Esta ventaja trae el beneficio de eliminar en gran medida los problemas de ambientes dados por configuraciones o instalaciones de software realizados de manera manual. Por lo tanto, tener un acceso restringido a los ambientes de pruebas de producción es una buena práctica recomendada incluso para los equipos de operaciones y desarrollo. De esta manera se desalienta la intervención manual de las personas para realizar cambios en caso de que se haya descubierto alguna falla.

Lo más probable que suele suceder cuando hay una falla es que se hagan cambios por fuera del ciclo formal y que a esos cambios no se les pueda dar un seguimiento, ya que fueron realizados de una manera no controlada. Por más que la persona que los haya realizado, los documente siempre está la posibilidad del error humano y por ende el cambio puede no estar registrado o no estar correctamente documentado.

Al tener una infraestructura versionada y gestionada a través de un pipeline automatizado, cualquier cambio que se quiera introducir se debe hacer mediante un mismo proceso formal y automatizado, que desencadenará la aplicación y la infraestructura a un estado saludablemente conocido. Además, la resolución de cambios de manera manual realizada por el personal de operaciones y/o desarrollo generalmente termina generando estrés innecesario en la organización.

3. Estrategias

Para el paradigma de Entrega Continua hay diferentes estrategias, técnicas y/o patrones que fueron adoptando las organizaciones de manera de habilitar la entrega software con alta calidad, reduciendo los tiempos de las caídas de los sistemas de cara al usuario, por lo cual los componentes principales de estas estrategias son disminuir el riesgo de implementación de un cambio de software y mejorar la experiencia de usuario.

Blue/Green

El patrón de implementación Blue/Green es bastante utilizado por organizaciones que adoptaron la práctica de Entrega Continua. Se utiliza para probar la aplicación antes de que sea desplegada al ambiente de producción donde es utilizada por todos los usuarios y provee de alta disponibilidad a la aplicación, es decir que reduce el posible tiempo de caída de una implementación. La estrategia consiste en crear un ambiente que sea idéntico al ambiente de producción. El ambiente actual de producción es rotulado como “Blue” y la nueva réplica del ambiente es rotulado como “Green”. Teniendo una réplica de todo el entorno productivo (Green), lo que se realiza es implementar la actualización o los cambios en el sistema en este ambiente y probar dichos cambios.

Hasta aquí no hay mayores cambios que si fuera probado en un ambiente de homologación o QA. La diferencia radica en que lo que se va a cambiar son configuraciones de red en el router (enrutador) para que el acceso de los usuarios se traslade de la red del ambiente “Blue” a la red del ambiente “Green”. En el caso de que se haya detectado alguna falla en el ambiente “Green”, la organización puede dar marcha atrás con el cambio (roll-back) y traspasar todo el tráfico de los usuarios al ambiente original (en este caso es Blue). Una vez que las pruebas son pasadas con éxito, el ambiente “Blue” pasa a ser el que tendrá las nuevas actualizaciones de nuevo software y se puede volver con el ciclo de implementación “Blue/Green”. Lo que nos permite este patrón es realizar pruebas en nuestro sistema en un ambiente idéntico al de producción y evita caídas en los deploy (zero-downtime).

Canary Release

Este patrón también es muy utilizado. La manera en que está estructurado es similar al patrón “Blue/Green” con la diferencia de que en este caso se puede controlar la cantidad de usuarios que pasan de un ambiente a otro, es decir que con el patrón Canary Release se puede en gran medida determinar la cantidad usuarios que estarán en el nuevo ambiente productivo.

Tal como se mencionó anteriormente, este patrón tiene una dinámica similar a “Blue/Green” por lo que la organización deberá tener al menos dos ambientes idénticos de producción para que se pueda cambiar la ruta de red, también mediante un router, y trasladar los usuarios gradualmente, a diferencia del patrón “Blue/Green” que deriva todo

el tráfico en un mismo momento. Una vez que la aplicación fue probada por una cantidad determinada de usuarios y la organización confirma que funciona correctamente se procede a ir transfiriendo más tráfico de usuarios al ambiente que contiene la implementación nueva. En el caso de que se detecte que algo no está funcionando correctamente se puede optar por volver los usuarios al ambiente original, reduciendo el impacto del fallo detectado sólo a los usuarios que estaban en el ambiente de la nueva versión.

Generalmente, el primer grupo de usuarios que se traslada al ambiente que contiene la nueva implementación son usuarios internos que pertenecen a la organización los cuales serán los primero que probarán la aplicación. Una vez probada, gradualmente se transfiere más usuarios al nuevo ambiente.

Estrategias con Microservicios, Contenedores y Arquitectura Inmutable

Las implementaciones con la práctica de entrega continua para una infraestructura o una arquitectura de microservicios varían de acuerdo a las necesidades de la organización. Lo que podemos mencionar aquí sin extender demasiado el tema y abarcándolo desde una perspectiva más genérica y generalista, es tomar los conceptos ya vistos anteriormente sobre pipeline e infraestructura versionada y aplicarlos al concepto de Entrega Continua.

La estrategia común de las organizaciones para este tipo de configuración es la de ir hacia una aplicación distribuida que esté construida en distintas partes desacopladas de manera que esas partes se encuentren aisladas unas con otras, las cuales tal vez contienen tecnologías y lenguajes de programación diferentes, de esta manera si ocurre alguna falla en uno de los servicios o contenedores no afecta en su totalidad el resto del sistema.

En algunas organizaciones donde es requerido un gran número de desarrolladores para un producto y en la cual se requiere de alta disponibilidad en el sistema (que no ocurran caídas de servicio), cada contenedor o servicio es elaborado por un equipo multidisciplinario que contiene cierta autonomía e incluye miembros que realizan las actividades de implementación y monitoreo en producción. Para que suceda este tipo de escenario, en el que distintas personas de distintas disciplinas y actividades colabore en un objetivo común que es el de dar un producto de calidad y brindar una buena experiencia de usuario, la organización debe ser capaz de adaptarse internamente para poder responder a las demandas del mercado, por ende, la organización debe ser capaz de diluir o disolver los silos funcionales en los que se encuentra estructurada.

La implementación de una infraestructura inmutable (a los cambios manuales) evita que se realicen cambios en los ambientes y/o en la aplicación de manera no controlada. Tener estas ventajas al implementar ya sea microservicios o una infraestructura que está tenida en cuenta en el pipeline de desarrollo es muy beneficioso, ya que las implementaciones poseen menor riesgo, costo y estrés, son más veloces, y los cambios son controlados y medidos. En el caso de que un servidor se cayera se podría levantar en un estado conocido a partir de unos pocos pasos. Además, se pueden implementar diferentes porciones del producto sin afectar la solución completa.

Feature Flag / Feature Toggle

Está técnica es una manera de desarrollar nuevos requerimientos que poseen una duración prolongada para su finalización y consiste en habilitar o deshabilitar la visibilidad del requerimiento al usuario mediante configuración. Esto significa que un equipo tendrá la capacidad de desarrollar nuevas funcionalidades integrándolas incrementalmente a través de un ciclo de Integración Continua y disponibilizarlas al usuario cuando se crea más conveniente (ya sea por una decisión de negocio o hasta esperar a terminar la funcionalidad) a través de parámetros simples de configuración. Además, esta técnica permite evitar hacer branch en el repositorio de versiones de código con lo que posibilita desarrollar requerimientos que demoren bastante tiempo en construirse.

Es importante diferenciar que la utilización de esta técnica cuando se utiliza para ocultar momentáneamente desarrollos que aún no están terminados, el código antiguo debe ser removido al momento de finalizar el desarrollo y las pruebas sobre el requerimiento, de esta manera se quita complejidad y pruebas automáticas innecesarias. Por el contrario, si la técnica se utiliza dado un requisito de negocio en el que una o varias funcionalidades se deben apagar o prender dependiendo algún criterio establecido, hay que ser cuidadoso en la manera de agregar este estilo de desarrollo ya que, como se mencionó anteriormente, agrega más complejidad a la aplicación lo que puede provocar que sea más costoso mantenerla.

Dark Launching & Dark Loading

Está técnica de implementación fue introducido por Facebook cuando estaban

desarrollando la funcionalidad del chat. La esencia de *Dark Launching* consiste en desarrollar y testear la infraestructura y la arquitectura necesaria para un determinado requerimiento en ambientes productivos que requieren alta disponibilidad y desempeño. Lo que se implementa en el ambiente de producción es la parte back end, es decir la parte no visible del usuario sobre el nuevo requerimiento y toda la arquitectura involucrada. Lo que habilita la utilización de este patrón es la posibilidad de probar la arquitectura y la infraestructura en un ambiente productivo sin la necesidad de que el usuario final lo sepa, generalmente mediante la desactivación del acceso en la interfaz de usuario, realizando una simulación de su interacción para aprovechar el tráfico de visitas del ambiente productivo, por este tipo de implementación es que se nombró como *Dark Launching* (lanzando en la oscuridad, implementando sin la visibilidad y conocimiento del usuario). No hay mejor ambientes para pruebas más efectivas que producción, por este motivo se prueba la infraestructura y la arquitectura en un ambiente productivo con miles de usuarios.

Por otro lado, el término *Dark Loading* (cargando en la oscuridad) es similar al de *Dark Launching* pero con el propósito de cargar y probar cambios en una nueva base de datos antes de que realizar el cambio de la que está en producción por la que contiene los nuevos cambios. Lo que permite este tipo de implementación para las bases de datos es no tener caídas en el servicio y probar los cambios realizados en la base de datos con anterioridad a la utilización por parte del usuario.

Bibliografía utilizada y sugerida

- Alex Williams. The Docker & Container Ecosystem. The New Stack.
- Jez Humble, David Farley. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. 1era edición. EE:UU: Addison Wesley; 2010
- Cycligent's whitepapper. Continuous Delivery for Design & Deployment.

Lo que vimos:

En la unidad recientemente vista se introdujo a las principales características de la práctica de Entrega Continua, las cuales consisten en conceptos que le competen a la organización ya que debe adecuar y alinear su estrategia tecnológica con la comercial, por lo cual la conformación de esta práctica no es una decisión directa del equipo de desarrollo.

Se mencionaron las principales características que posee la infraestructura versionada en conjunto con la visión de alcanzar una arquitectura inmutable en un pipeline de desarrollo. Además, se hizo mención de las principales estrategias de implementación para la práctica de Entrega Continua, identificando ventajas y desventajas.

Lo que viene:

En la siguiente unidad se realizará una reseña por algunas empresas que están inmersas en el movimiento DevOps y los cuales son tomadas como casos de éxito dado la capacidad y velocidad de adaptación que poseen con respecto a la evolución técnica y comercial.