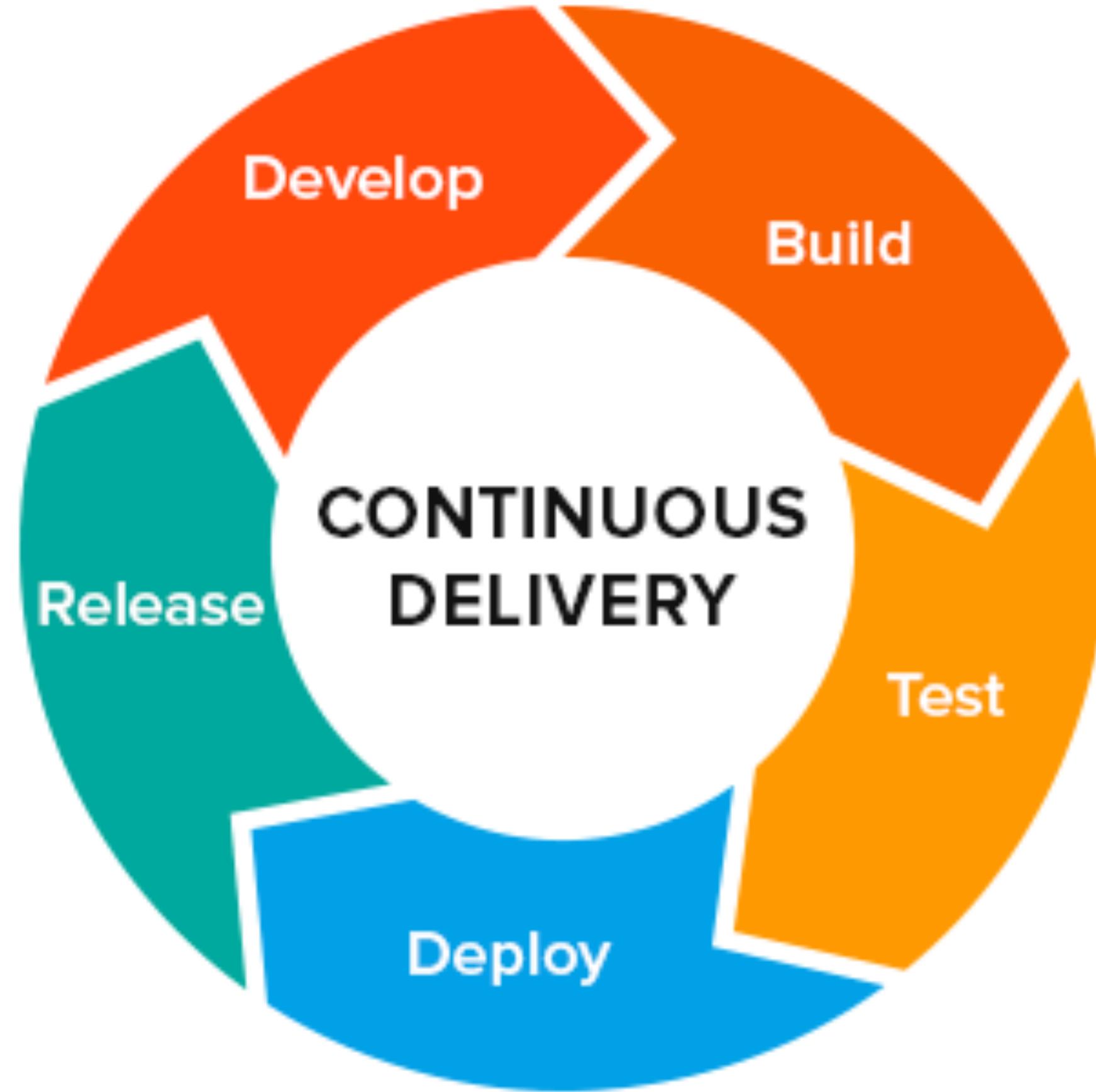


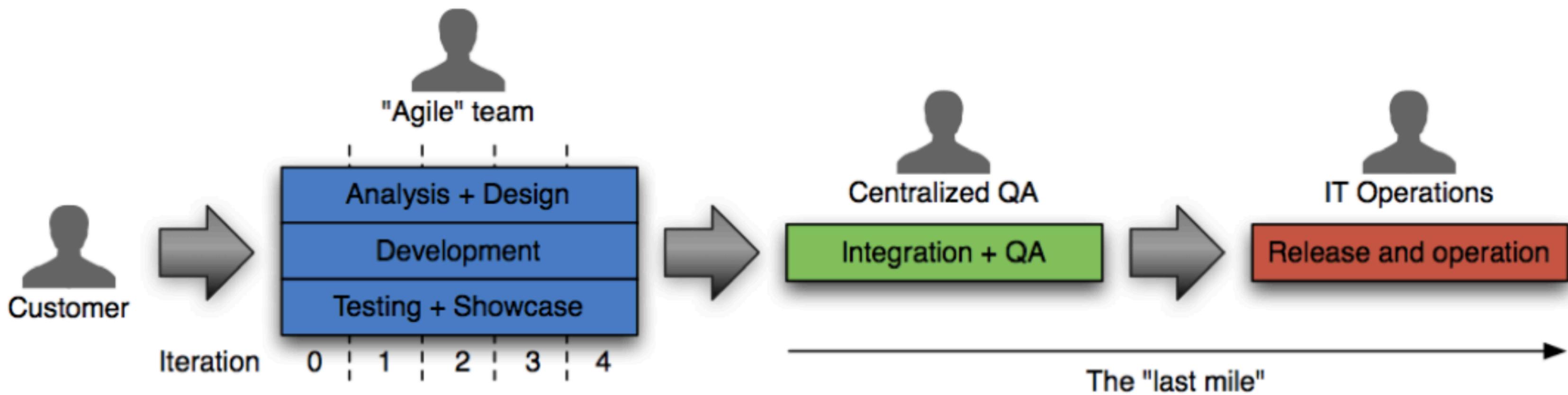
MADS

S10: Continuous Delivery

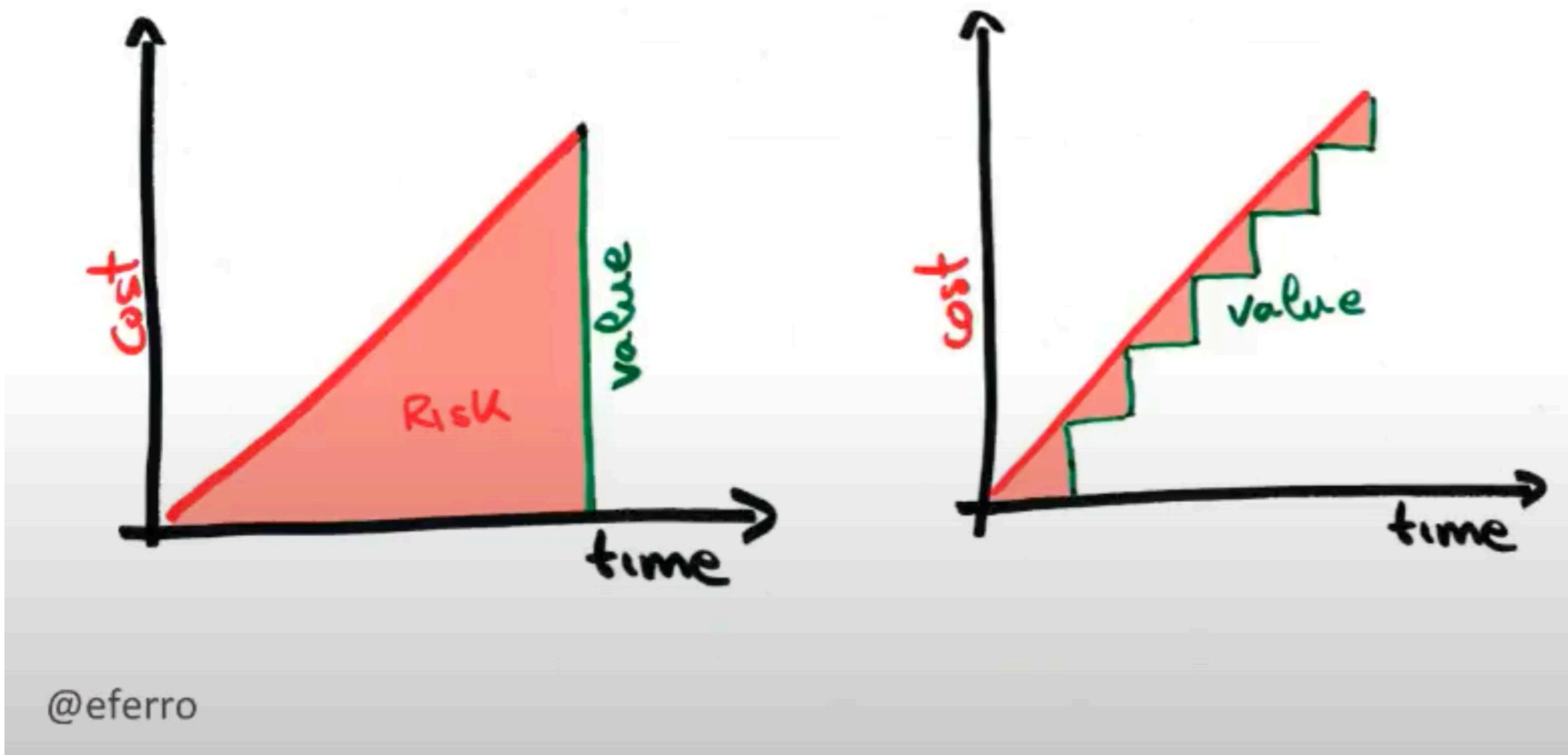
(Bloque 3 - Continuous Delivery)



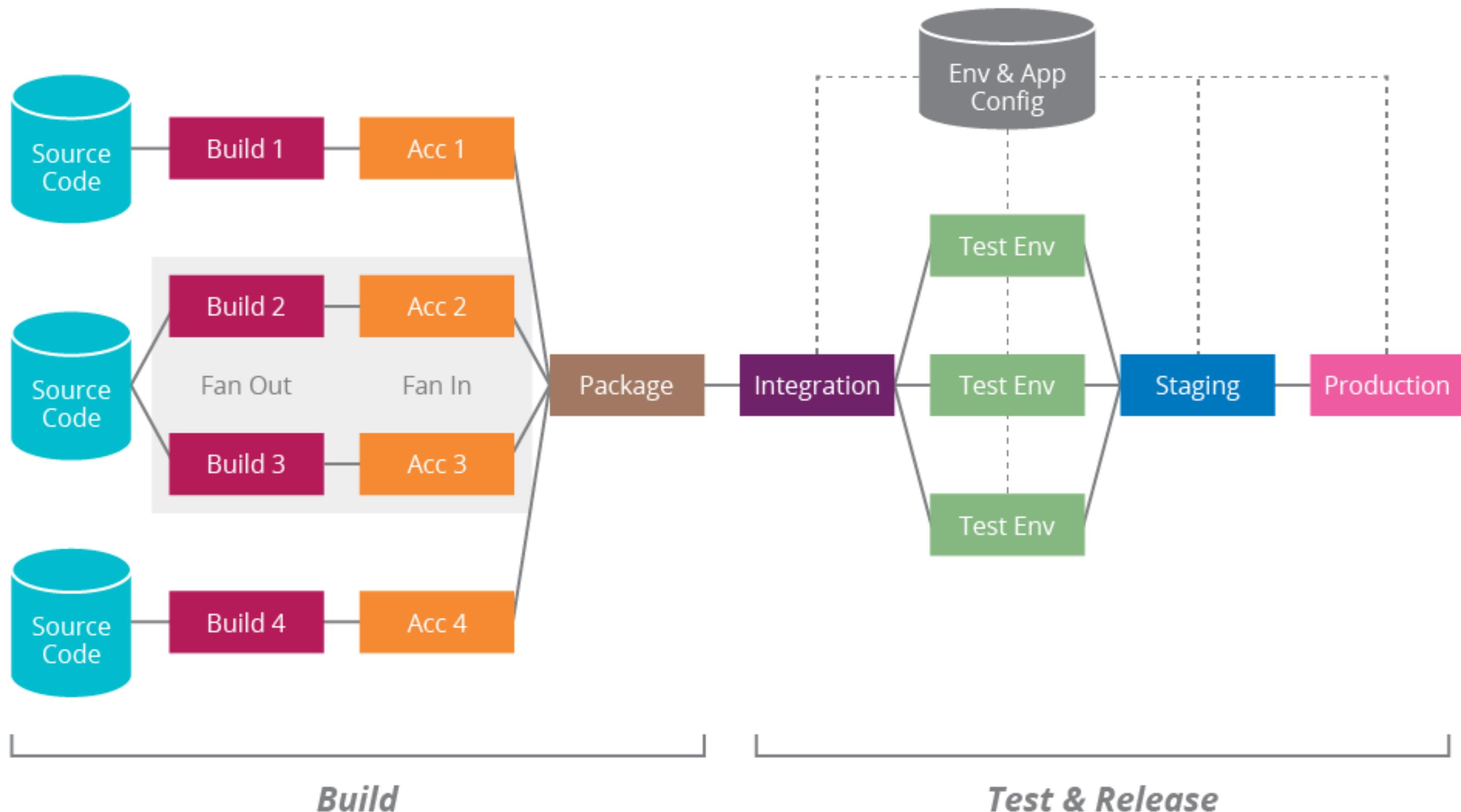
La última milla



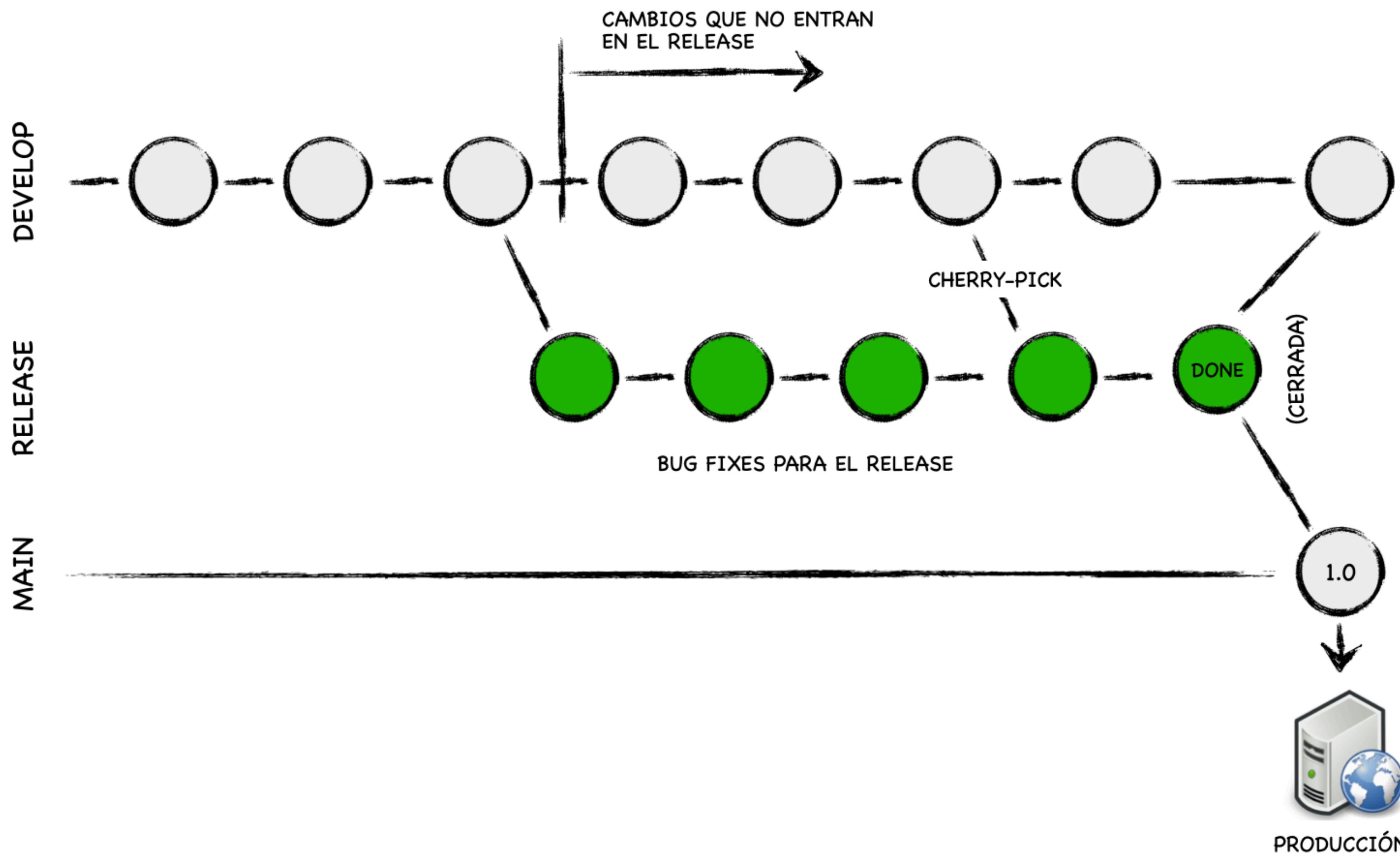
Agilidad: Gestión riesgo / Adaptabilidad



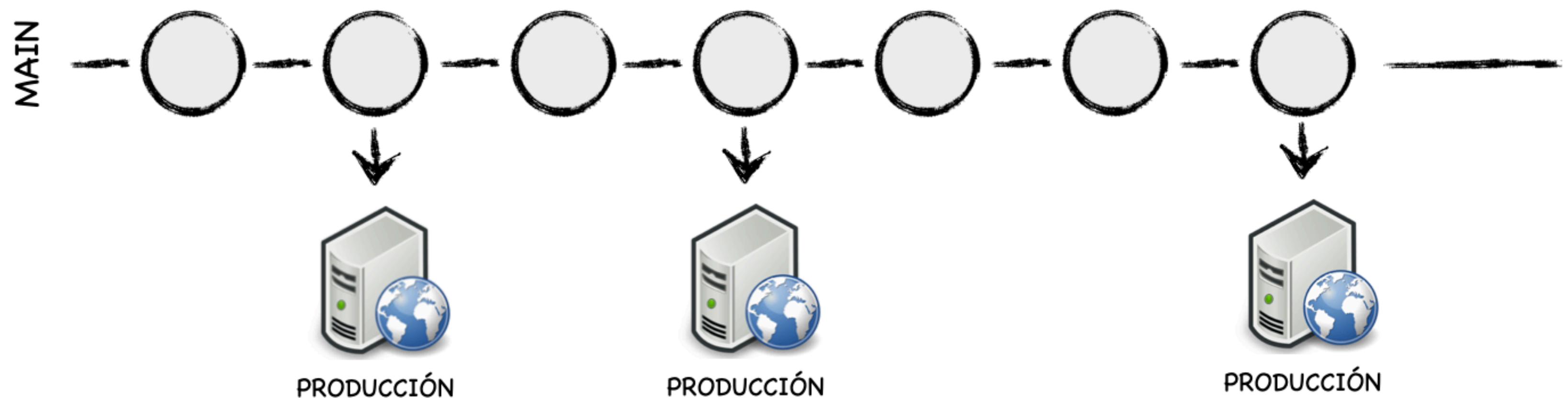
Tubería de despliegue



Enfoque clásico del lanzamiento de versiones

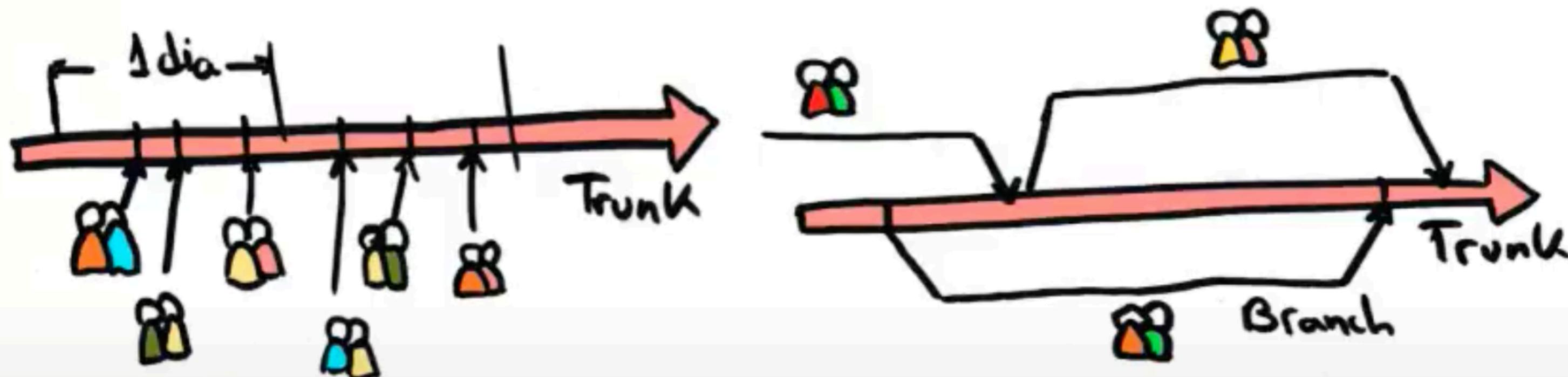


Entrega continua

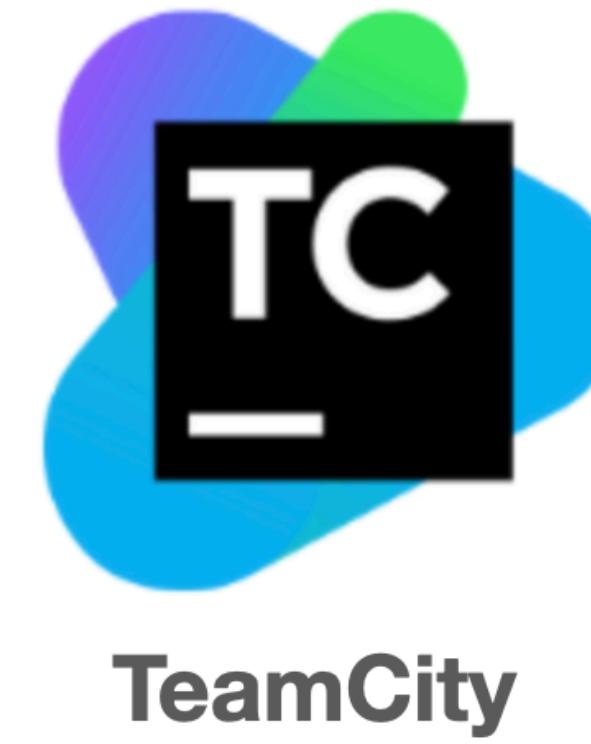


Prácticas de commits en IC

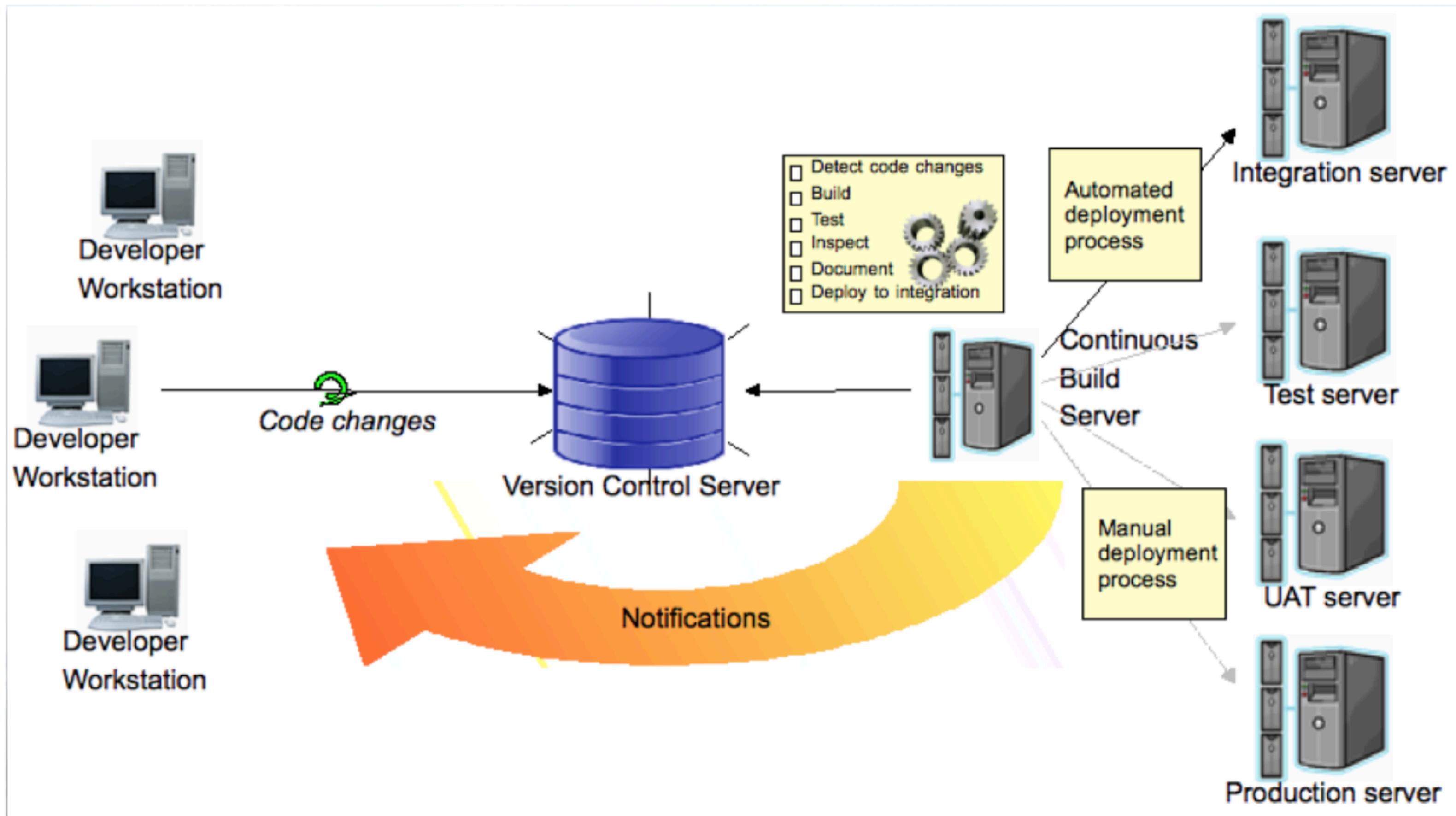
Trunk based vs Feature branches



Herramientas de IC



Sistema de Integración Continua



Panel de control en Jenkins

Jenkins search log in

Jenkins core ENABLE AUTO REFRESH

People Build History Project Relationship Check File Fingerprint Disk usage We Need Beer

Build Queue No builds in the queue.

Build Executor Status

#	Status
	master
1	Building infra update center experimental #389
2	Idle remote-slave-3 (offline) remote-slave-6
1	Idle
2	Building jenkins main trunk #3074

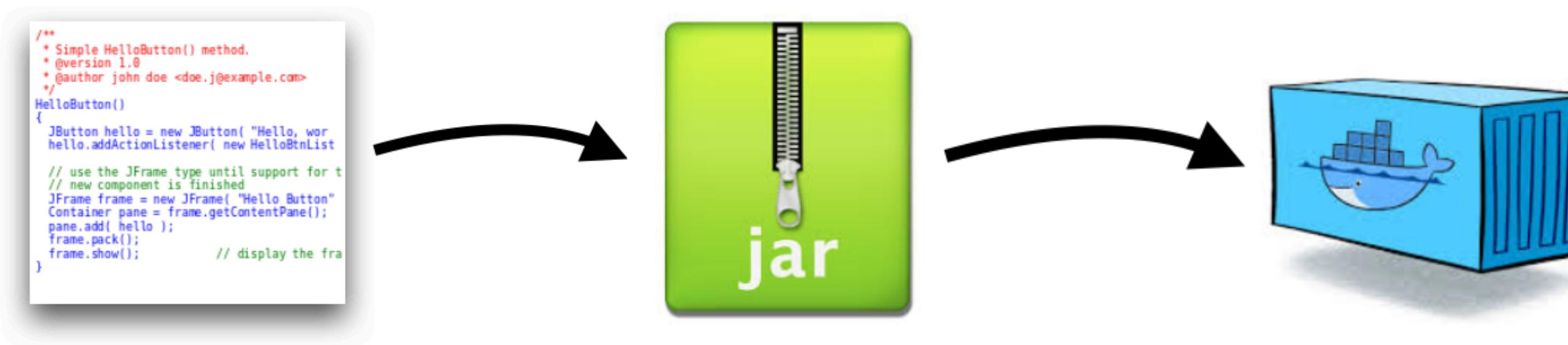
All All Disabled All Failed All Unstable Infrastructure Jenkins core Libraries Other Projects

S	W	Name ↓	Last Success	Last Failure	Last Duration	LC
Red	Sunny	jenkins_its_branch	4 days 4 hr - #127	3 days 16 hr - #128	53 min	
Blue	Cloudy	jenkins_main_maven-3.1.0	4 mo 2 days - #7	4 mo 2 days - #6	1 hr 11 min	
Blue	Sunny	jenkins_main_trunk	1 day 7 hr - #3073	11 days - #3035	1 hr 7 min	
Blue	Sunny	jenkins_pom	3 days 13 hr - #212	N/A	45 sec	
Yellow	Sunny	jenkins_rc_branch	3 days 16 hr - #381	24 days - #373	2 hr 35 min	
Red	Cloudy	jenkins_ui-changes_branch	1 yr 5 mo - #32	1 yr 1 mo - #33	4 min 55 sec	
Blue	Rainy	remoting	5 mo 12 days - #4	5 mo 12 days - #3	6 min 19 sec	

Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

Generación del ejecutable



	swift-5.3-DEVELOPMENT-SNAPSHOT-2020-11-11-a-centos8.tar.gz		
	Versión	Fecha build	SO

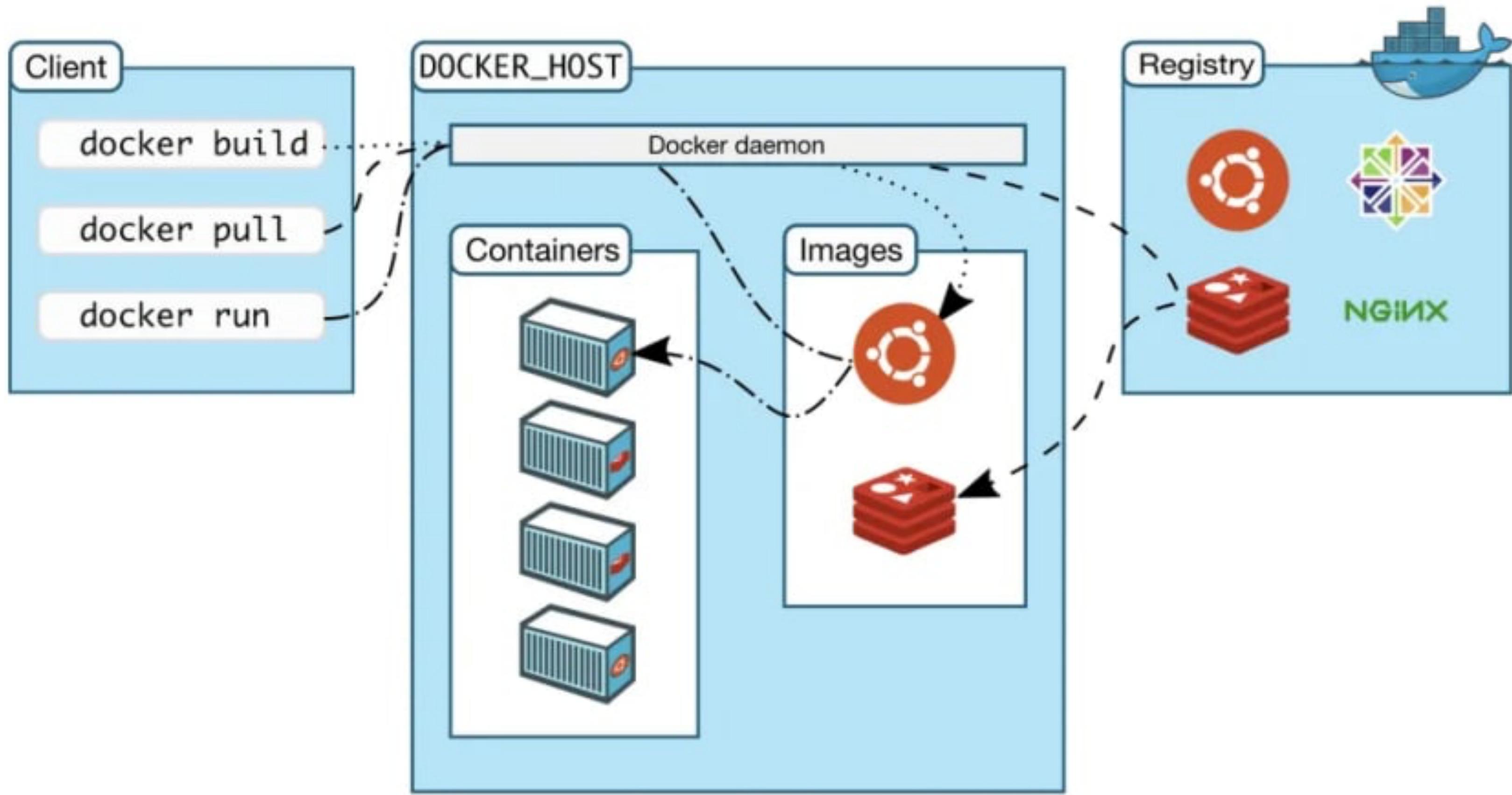
Principios y prácticas de IC (1)

- Desarrollo de código
 - El sistema debe siempre poder ser construido (build) y probado con éxito.
 - Todo el mundo hace merge de los cambios con frecuencia.
 - Después de cada commit, el sistema se integra inmediata y automáticamente.
 - Se desarrolla el sistema en pequeños incrementos.
- Testing
 - Los desarrolladores prueban su código en sus espacios de trabajo privados.
 - Después mezclan los cambios en el repositorio.

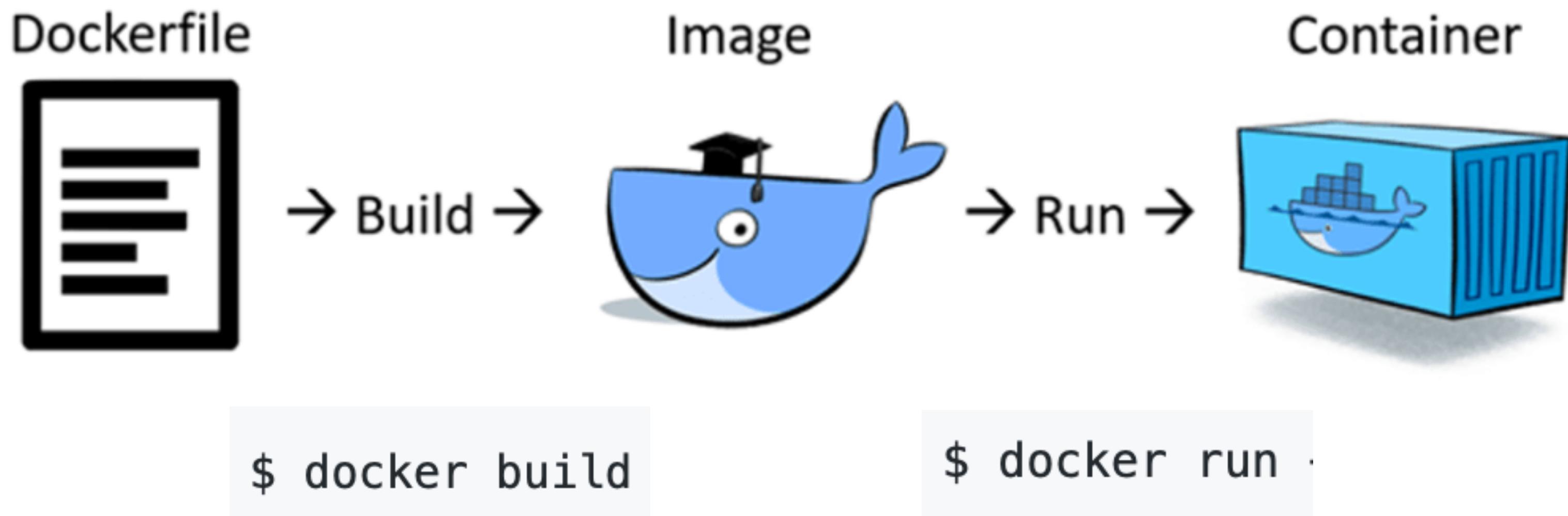
Principios y prácticas de IC (2)

- Servidor de integración continua (CI server):
 - Monitoriza el repositorio y comprueba los cambios cuando ocurren.
 - Construye el sistema y ejecuta las pruebas unitarias y de integración.
 - Informa al equipo de la construcción con éxito o de los fallos.
- Errores en los build
 - El equipo arregla el problema lo antes posible.
 - Continuar para integrar y probar continuamente durante todo el proyecto.
- El último ejecutable compilado debe estar fácilmente disponible
 - El resultado de la compilación debe ser un artefacto ejecutable disponible para desplegar en distintos entornos, incluso en producción.

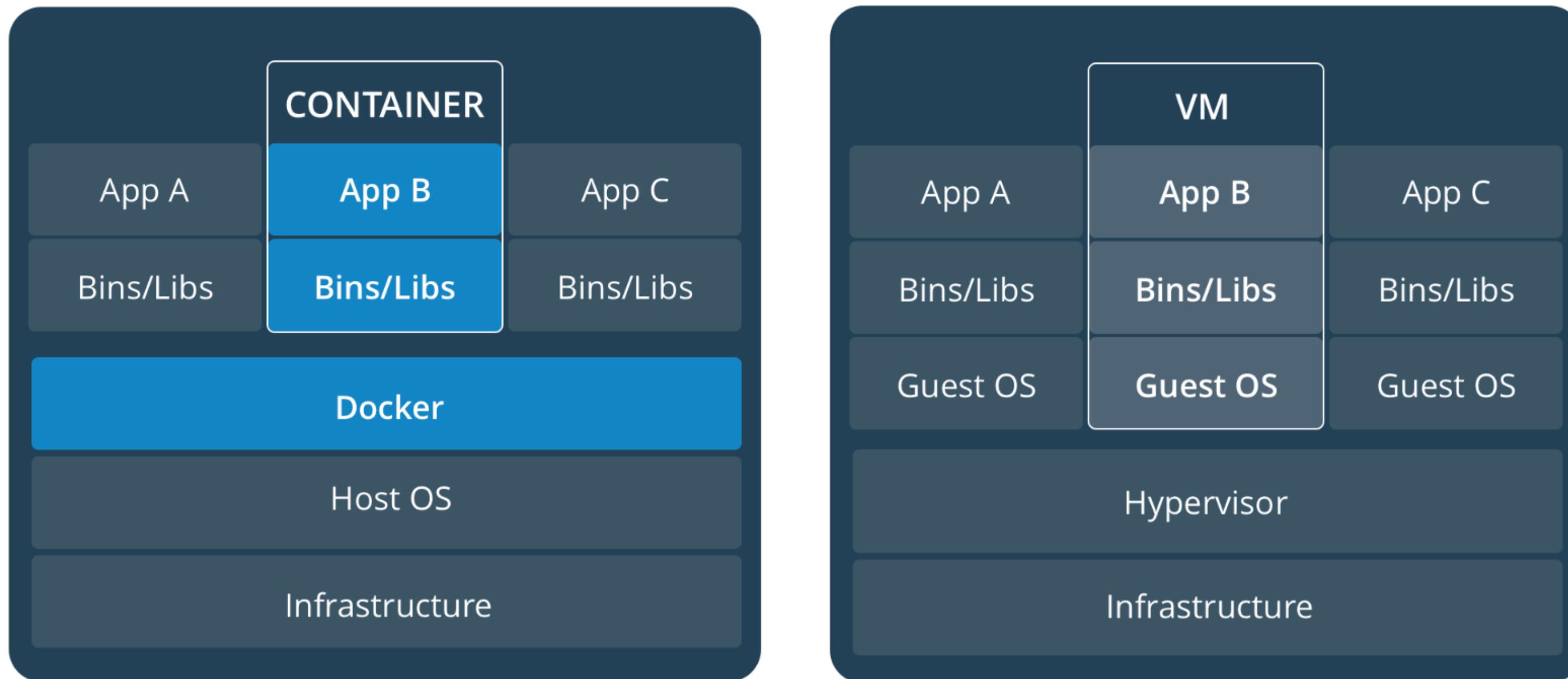
Docker



Imágenes y contenedores



Contenedores ligeros



Dockerfile

Imagen base

```
FROM ubuntu:14.04
```

```
# install cowsay, and move the "default.cow" out of  
# the way so we can overwrite it with "docker.cow"
```

```
RUN apt-get update && apt-get install -y cowsay \  
    --no-install-recommends && rm -rf /var/lib/apt/lists/* \  
    && mv /usr/share/cowsay/cows/default.cow \  
        /usr/share/cowsay/cows/orig-default.cow
```

```
# "cowsay" installs to /usr/games
```

```
ENV PATH $PATH:/usr/games
```

```
COPY docker.cow /usr/share/cowsay/cows/
```

```
RUN ln -sv /usr/share/cowsay/cows/docker.cow \  
    /usr/share/cowsay/cows/default.cow
```

```
CMD [ "cowsay" ]
```

Comandos que se ejecutan al construir la imagen

Comando al ejecutar el contenedor

Demo de Docker

Entornos de despliegue

- **Local:** ordenador del desarrollador. Se ejecutan tests unitarios de la característica que se está desarrollando.
- **Desarrollo/Trunk/Master:** ordenador de integración continua conectado a la rama de desarrollo en el que se ejecutan todos los tests unitarios continuamente.
- **Integración:** Entorno en el que se sustituyen los mocks y bases de datos de memoria por servicios reales, aunque con copias parciales de los datos de producción.
- **Test/QA:** Entornos en los que se realizan pruebas funcionales y de interfaz de usuario. Pueden ser manuales.
- **Stage/Preproducción:** Entorno idéntico al de producción en el que se hace la última validación de la nueva versión a desplegar a producción. Copia de la base de datos de producción y con servidores similares a los de producción, para poder comprobar rendimiento.
- **Producción:** Entorno que usan los clientes reales de la aplicación.

Ejemplo de configuración en Spring Boot

Fichero *application-production.properties*

```
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
DB_USER=mads
DB_PASSWD=mads

spring.datasource.url=jdbc:postgresql://${POSTGRES_HOST}:${POSTGRES_PORT}/mads
spring.datasource.username=${DB_USER}
spring.datasource.password=${DB_PASSWD}
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQL9Dialect
spring.datasource.initialization-mode=never
```

Script de ejecución

```
$ export PROFILE=production
$ export DB_IP=3316
$ export DB_USER=mads
$ export DB_PASSWD=mads
$ java -Dspring.profiles.active=$PROFILE -Ddb_ip=$DB_IP -Ddb_user=$DB_USER \
-Ddb_passwd=$DB_PASSWD -Dlogging=$LOGGING -jar target/mads-todolist-inicial-0.0.1-SNAPSHOT.jar
```

Ejemplo de configuración en Docker

Dockerfile

```
FROM alpine
ENV saludo="Hola, mundo!"
CMD echo $saludo
```

```
$ docker build -t saludo .
$ docker run saludo
Hola, mundo!
```

```
$ docker run -e "saludo=¿Qué tal estás?" saludo
¿Qué tal estás?
```

propiedades.txt

```
saludo=¿Qué passsa, colega?
```

```
$ docker run --env-file=propiedades.txt saludo
¿Qué passsa, colega?
```

Imagen Docker con Spring Boot configurable

Ficheros de propiedades con variables para configurar la imagen Spring Boot

Fichero `application-postgres.properties` :

```
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
DB_USER=mads
DB_PASSWD=mads
spring.datasource.url=jdbc:postgresql://${POSTGRES_HOST}:${POSTGRES_PORT}/mads
spring.datasource.username=${DB_USER}
spring.datasource.password=${DB_PASSWD}
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQL9Dialect
spring.datasource.initialization-mode=never
spring.jpa.hibernate.ddl-auto=validate
```

Fichero `application-postgres-prod.properties` :

```
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
DB_USER=mads
DB_PASSWD=mads
spring.datasource.url=jdbc:postgresql://${POSTGRES_HOST}:${POSTGRES_PORT}/mads
spring.datasource.username=${DB_USER}
spring.datasource.password=${DB_PASSWD}
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQL9Dialect
spring.datasource.initialization-mode=never
spring.jpa.hibernate.ddl-auto=validate
```

Dockerfile para pasar parámetros

Fichero `Dockerfile` :

```
FROM openjdk:8-jdk-alpine
COPY target/*.jar app.jar
ENTRYPOINT ["sh", "-c", "java -jar /app.jar ${0} ${@}"]
```

Comando para lanzar la imagen

```
$ docker run --rm domingogallardo/mads-todolist:latest
--spring.profiles.active=postgres-prod --POSTGRES_HOST=postgres
```

Gestión de la BD de producción

- La base de datos de producción es uno de los activos más importante de nuestra aplicación.
- Debemos definir políticas de respaldo y de control de cambios para evitar que se produzca cualquier pérdida de información.
- Cada versión nueva de la aplicación va a necesitar actualizar el esquema de datos de la BD, manteniendo la integridad de los datos ya existentes.

The screenshot shows a dark-themed website header for martinFowler.com with navigation links for Refactoring, Agile, Architecture, About, and Thoughtwork. Below the header, the title 'Evolutionary Database Design' is displayed in large, bold, black font. A detailed paragraph in English explains the evolution of database design techniques over a decade, emphasizing continuous integration and automated refactoring. The date 'May 2016' is visible above a horizontal line. To the left of the line is a small portrait photo of Pramod Sadalage. To the right is a 'CONTENTS' section listing several key points: Jen implements a new story, Dealing with Change, Limitations, The Practices, DBAs collaborate closely with developers, All database artifacts are version controlled with application code, All database changes are migrations, Everybody gets their own database instance, Developers continuously integrate database changes, A database consists of schema and data, and All database changes are database refactorings.

martinFowler.com

Refactoring Agile Architecture About Thoughtwork

Evolutionary Database Design

Over the last decade we've developed and refined a number of techniques that allow a database design to evolve as an application develops. This is a very important capability for agile methodologies. The techniques rely on applying continuous integration and automated refactoring to database development, together with a close collaboration between DBAs and application developers. The techniques work in both pre-production and released systems, in green field projects as well as legacy systems.

May 2016

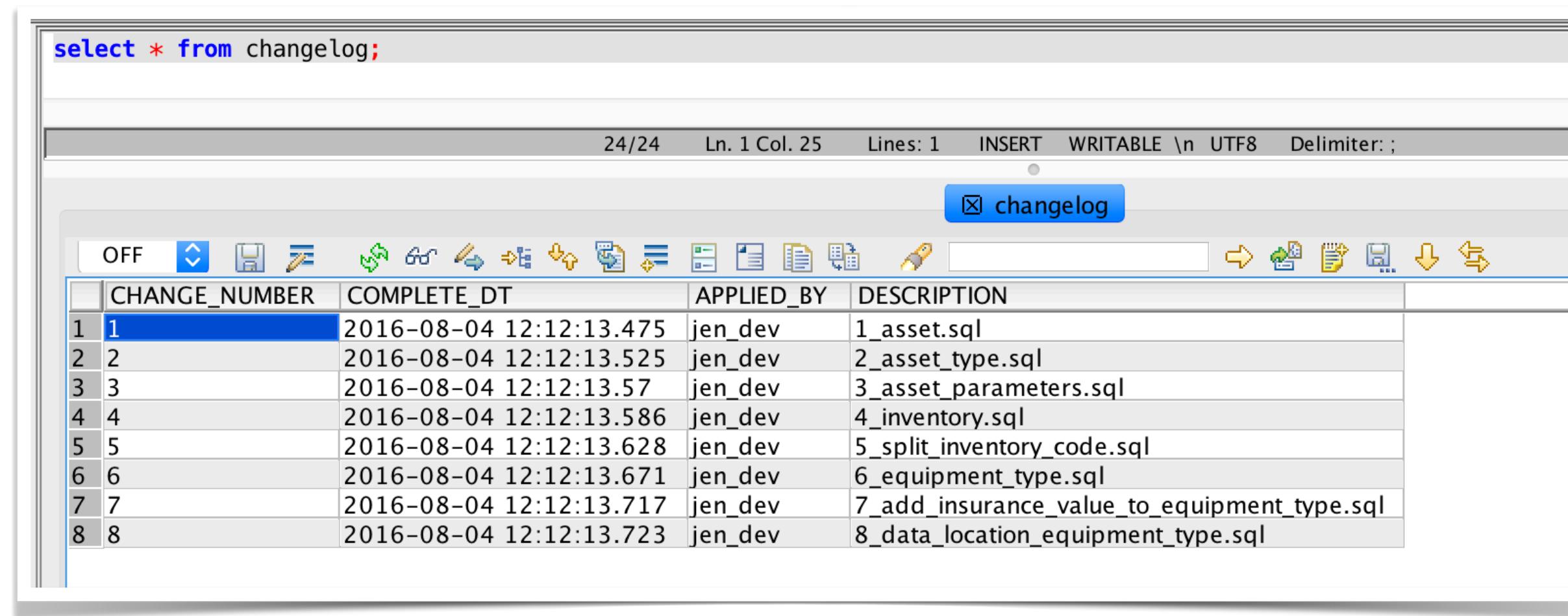
 Pramod Sadalage

Pramod developed the original techniques of evolutionary database design and database refactoring used by Thoughtworks in 2000. Since then he has worked with many clients world-wide using and developing these techniques, training and consulting them on how to do it.

CONTENTS

- Jen implements a new story
- Dealing with Change
- Limitations
- The Practices
 - DBAs collaborate closely with developers
 - All database artifacts are version controlled with application code
 - All database changes are migrations
 - Everybody gets their own database instance
 - Developers continuously integrate database changes
 - A database consists of schema and data
 - All database changes are database refactorings

Diseño evolutivo de la base de datos



The screenshot shows a MySQL Workbench interface. In the top-left pane, there is a SQL editor window containing the following query:

```
select * from changelog;
```

Below the editor, the status bar displays: 24/24, Ln. 1 Col. 25, Lines: 1, INSERT, WRITABLE, \n, UTF8, Delimiter: ;.

The main area is a grid-based table viewer titled "changelog". The table has four columns: CHANGE_NUMBER, COMPLETE_DT, APPLIED_BY, and DESCRIPTION. The data is as follows:

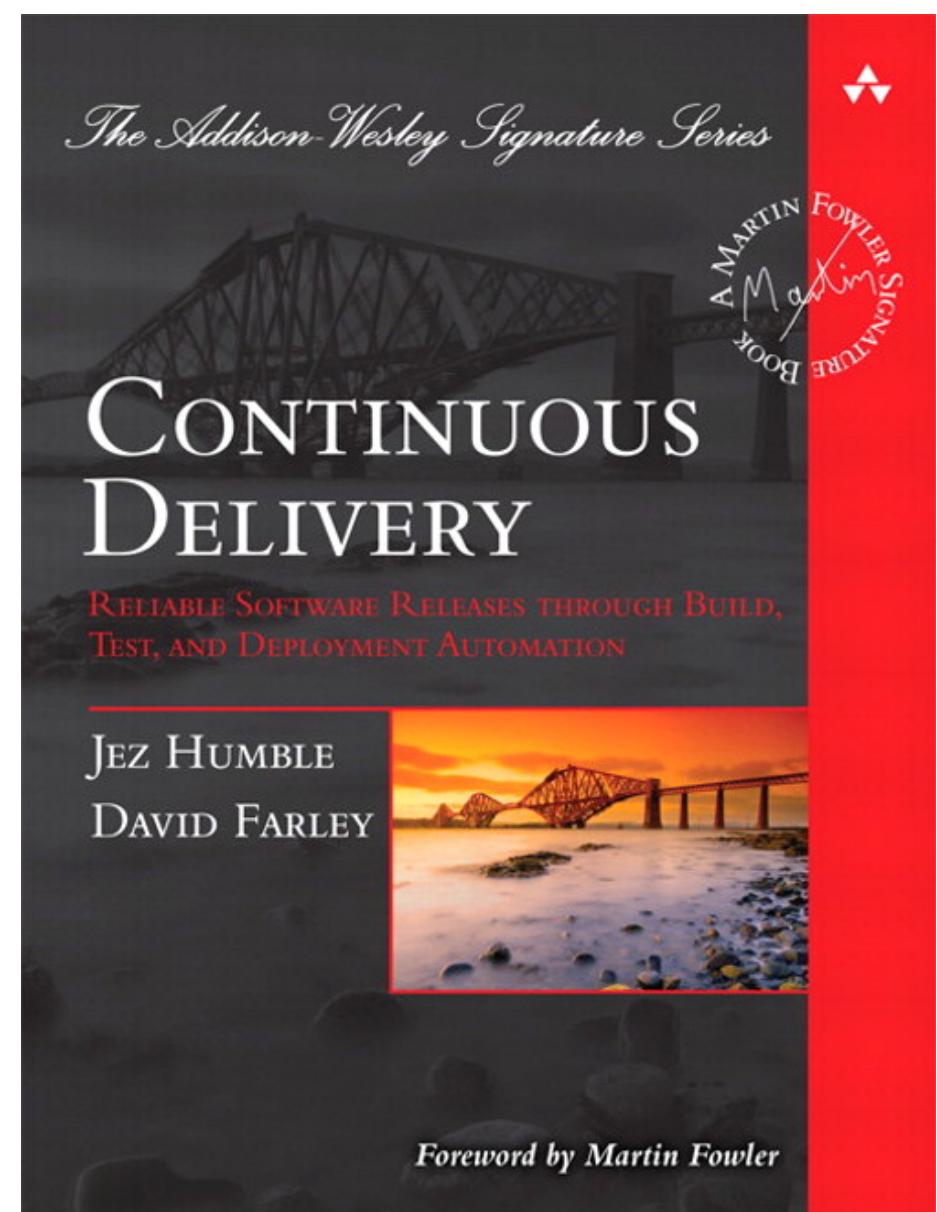
	CHANGE_NUMBER	COMPLETE_DT	APPLIED_BY	DESCRIPTION
1	1	2016-08-04 12:12:13.475	jen_dev	1_asset.sql
2	2	2016-08-04 12:12:13.525	jen_dev	2_asset_type.sql
3	3	2016-08-04 12:12:13.57	jen_dev	3_asset_parameters.sql
4	4	2016-08-04 12:12:13.586	jen_dev	4_inventory.sql
5	5	2016-08-04 12:12:13.628	jen_dev	5_split_inventory_code.sql
6	6	2016-08-04 12:12:13.671	jen_dev	6_equipment_type.sql
7	7	2016-08-04 12:12:13.717	jen_dev	7_add_insurance_value_to_equipment_type.sql
8	8	2016-08-04 12:12:13.723	jen_dev	8_data_location_equipment_type.sql

- Usar herramientas como Flyway o Liquibase (para Java)
- Cualquier migración debe identificarse con un script y debe poder ser reversible, de forma que podamos volver al estado anterior en el que se encontraba la base de datos.
- Mantener un registro histórico con todas las migraciones realizadas a la base de datos
- Cada desarrollador tiene su propia base de datos para desarrollo y prueba, y puede tener un esquema distinto al de la base de datos de producción.
- Automatizar la creación de esquemas de bases de datos usando scripts

Entrega continua

Puesta en producción:

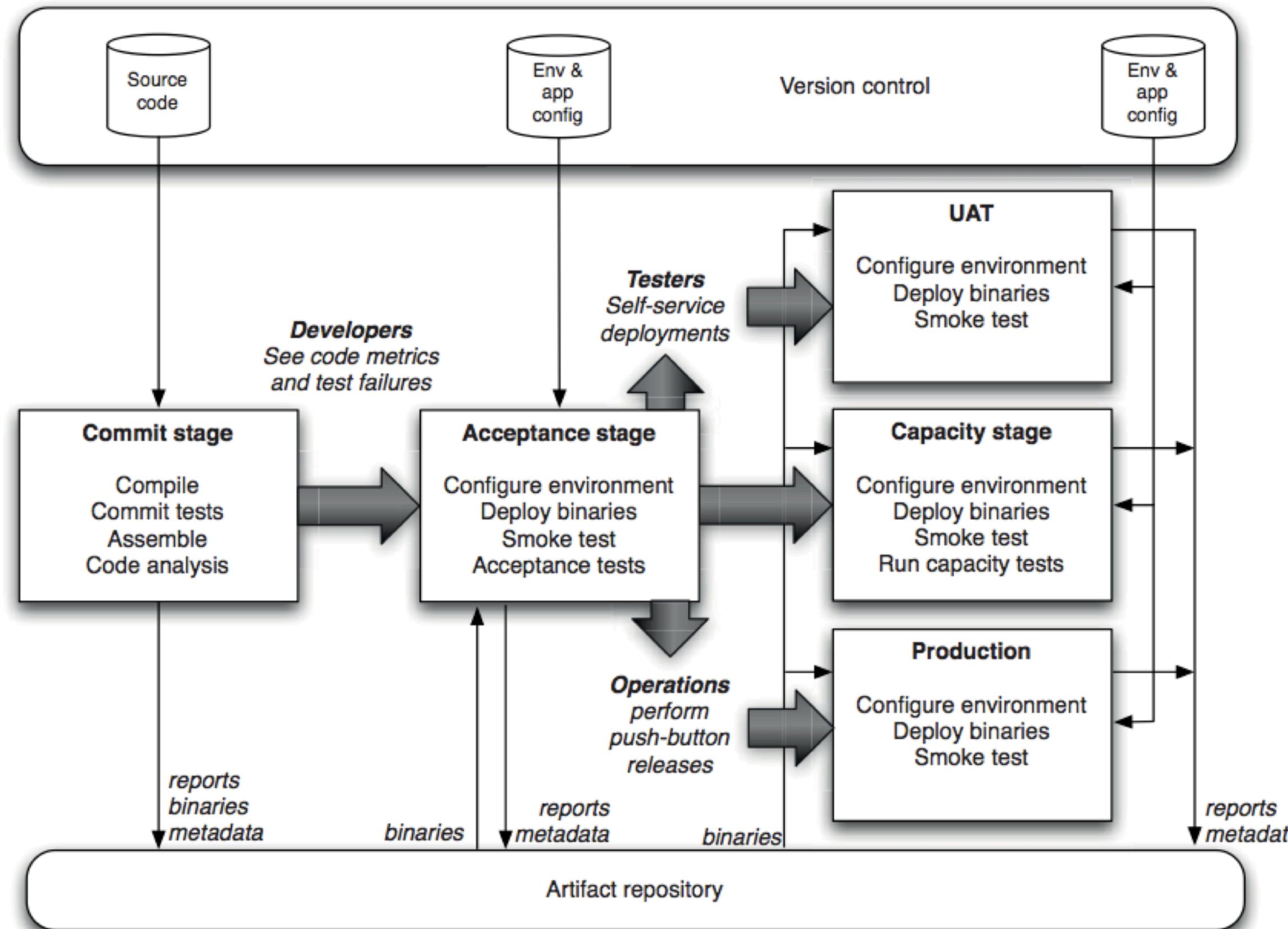
- Poco arriesgada
- Frecuente
- Barata
- Rápida
- Predecible
- Reproducible



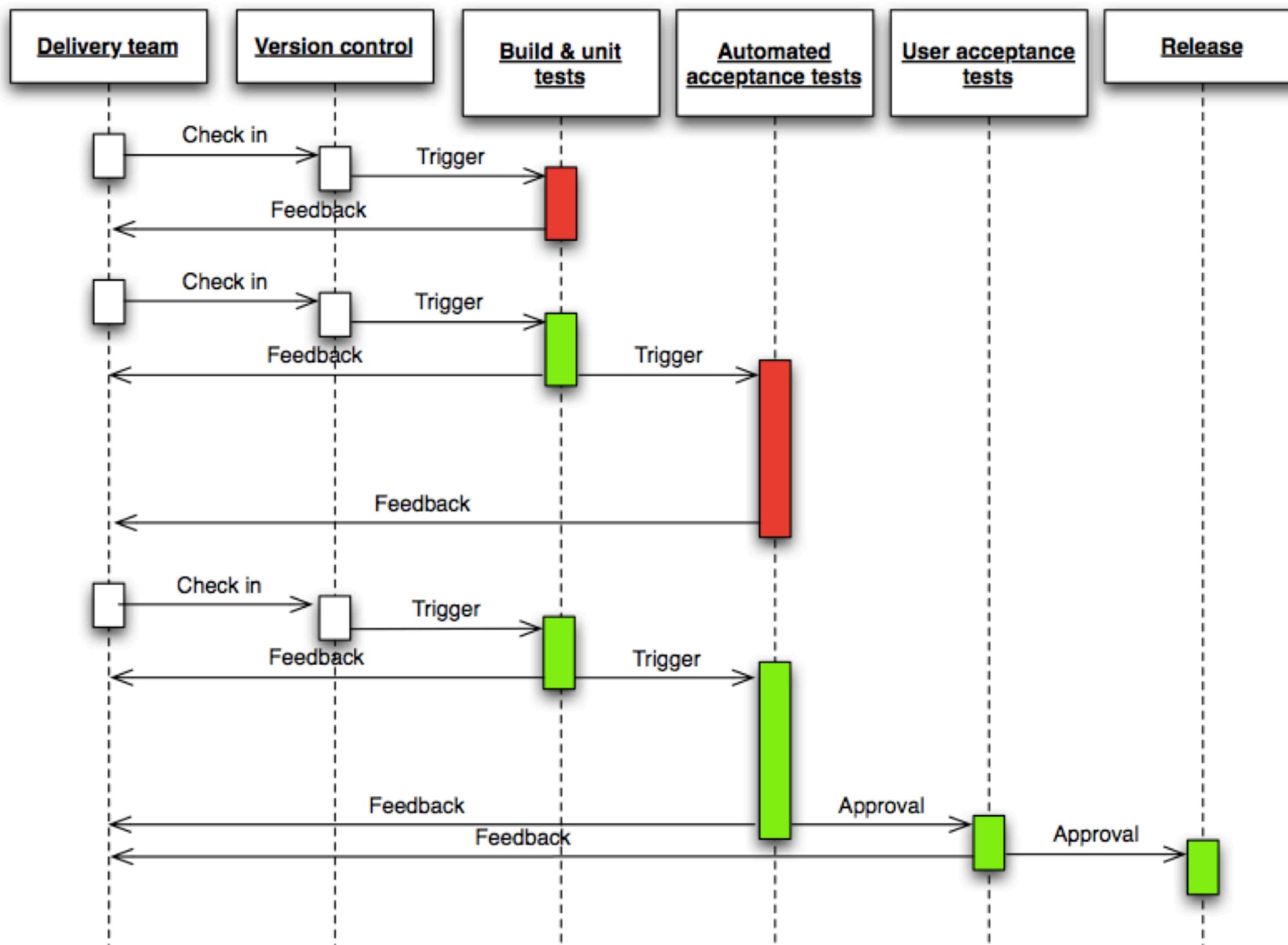
Técnicas

- Pequeños cambios que se despliegan continuamente
- Todos los builds son candidatos al release
- Todo en el control de versiones (se debe poder probar cualquier release)
- Tuberías de despliegue (deployment pipelines)
- Integración continua: automatización de builds, tests, despliegues, entornos

Tubería de despliegue



Flujo



Panel de control

Jenkins search Trevor Quinn | log out

Jenkins Build Pipeline DISABLE AUTO REFRESH

Build Pipeline

Run History Configure Add Step Delete Manage

The Jenkins Build Pipeline interface displays four parallel pipelines (87, 86, 85, 84) with stages: Commit, Acceptance, UAT, and Production. Each pipeline consists of five boxes: Pipeline, Commit, Acceptance, UAT, and Production. Pipeline boxes are grey, Commit boxes are green, Acceptance boxes are red, UAT boxes are light blue, and Production boxes are cyan. Arrows indicate the flow from Commit to Acceptance, Acceptance to UAT, and UAT to Production. Pipeline #87 is currently in the UAT stage. Pipelines #86, #85, and #84 have completed their respective stages.

Pipeline	Commit	Acceptance	UAT	Production
#87	#87 Commit Dec 18, 2014 2:07:08 PM 24 sec developer	#102 Acceptance Dec 18, 2014 2:12:43 PM 32 sec	#48 UAT Dec 18, 2014 2:13:23 PM 0.23 sec tester	Production
#86	#86 Commit Dec 3, 2014 9:45:11 AM 13 sec developer	#99 Acceptance Dec 3, 2014 10:13:16 AM 32 sec	#45 UAT Dec 3, 2014 10:15:01 AM 31 sec	#28 Production Dec 3, 2014 10:16:41 AM 38 sec
#85	#85 Commit Dec 3, 2014 9:42:11 AM 12 sec developer	#97 Acceptance Dec 3, 2014 9:42:31 AM 50 sec	UAT	Production
#84	#84 Commit Dec 3, 2014 9:39:31 AM 9.6 sec developer	Acceptance	UAT	Production

CartoDB

Continuous Integration at CartoDB

JUAN IGNACIO SÁNCHEZ

Software Craftsmanship Madrid



Pequeños cambios continuos

The image shows a Jenkins project dashboard for the 'Deploy CartoDB to Production' project. The dashboard includes a sidebar with links for Back to Dashboard, Status, Changes, and Workspace. The main area displays the 'Project Deploy CartoDB to Production' page with sections for Workspace, Recent Changes, Subprojects, and Permalinks. A large text overlay on the right side of the dashboard states '15 / week peak: 7'.

Project Deploy CartoDB to Production

Workspace

Recent Changes

Subprojects

Static

Purge Varnish(non-blocking)

Static

Purge Varnish(non-blocking)

Permalinks

- Last build (#1627), 2 hr 41 min ago
- Last stable build (#1627), 2 hr 41 min ago
- Last successful build (#1627), 2 hr 41 min ago
- Last failed build (#1591), 24 days ago
- Last unsuccessful build (#1591), 24 days ago

15 / week
peak: 7

Build History

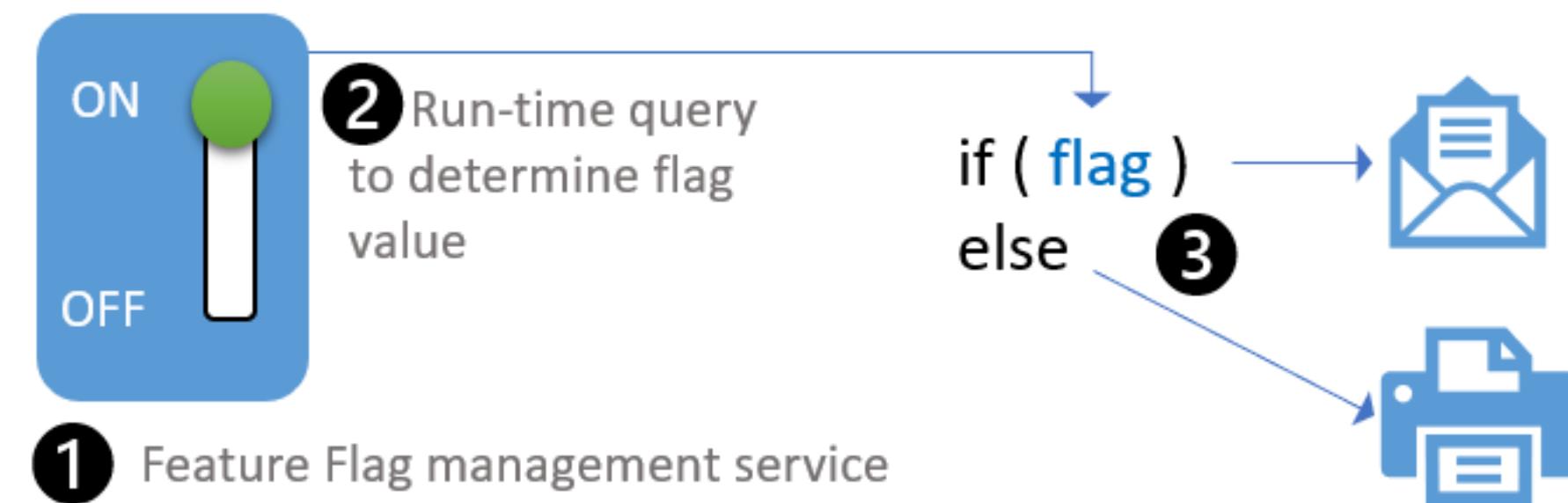
Build #	Date
#1627	Nov 27, 2015 1:45 PM
#1626	Nov 27, 2015 10:13 AM
#1625	Nov 26, 2015 10:50 AM
#1624	Nov 25, 2015 1:47 PM
#1623	Nov 25, 2015 10:48 AM
#1622	Nov 24, 2015 3:53 PM
#1621	Nov 24, 2015 3:23 PM
#1620	Nov 24, 2015 2:59 PM
#1619	Nov 24, 2015 2:32 PM
#1618	Nov 24, 2015 12:03 PM
#1617	Nov 24, 2015 10:52 AM
#1616	Nov 24, 2015 9:05 AM
#1615	Nov 23, 2015 1:13 PM
#1614	Nov 23, 2015 11:44 AM
#1613	Nov 23, 2015 9:03 AM

¿Y los cambios grandes?

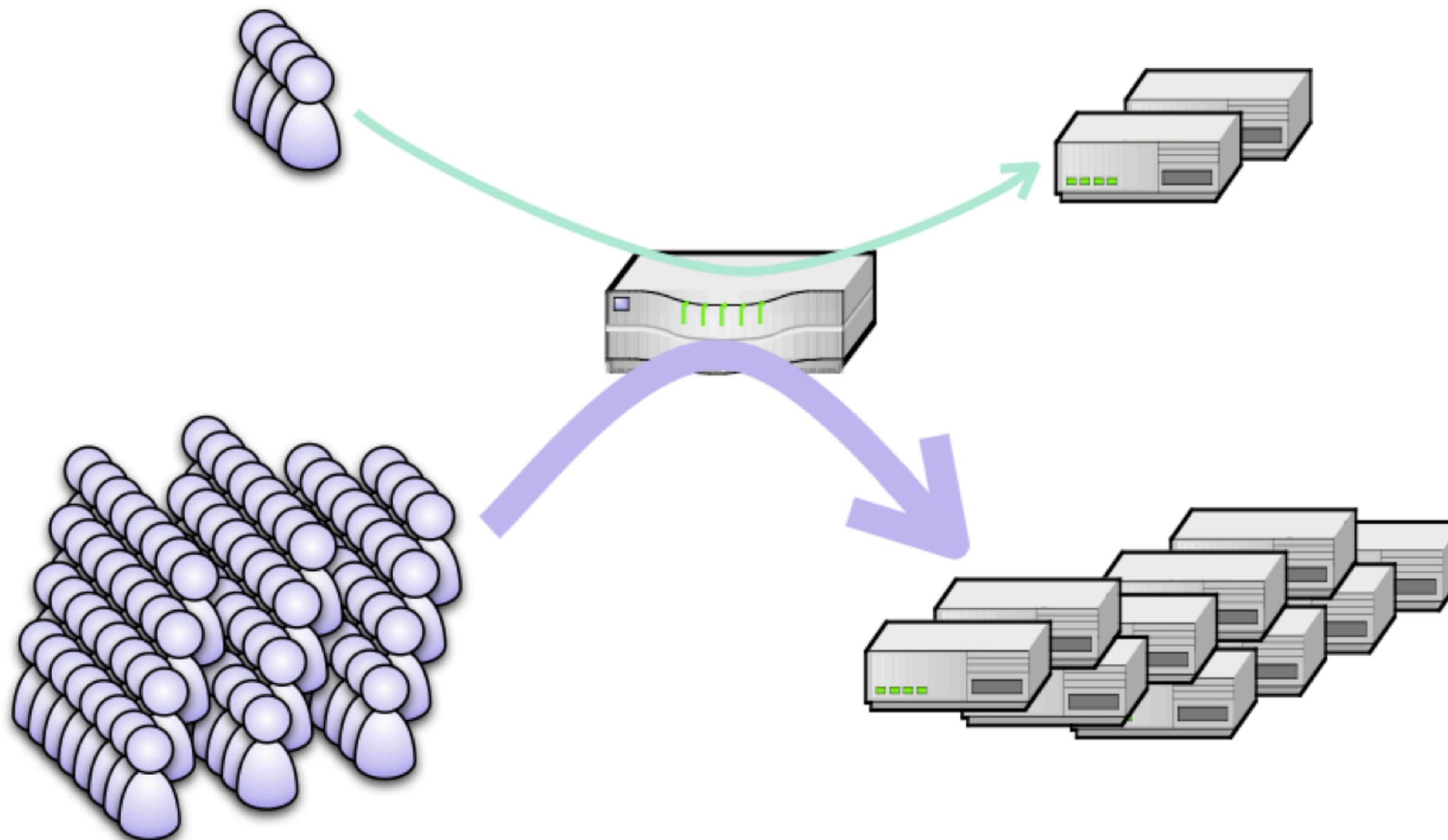
- Codificación y prueba de las pequeñas funcionalidades por separado.
- Buen diseño de código, por ejemplo seleccionar una implementación concreta utilizando interfaces y factorías, pero dejar la estructura lista para introducir futuros cambios.
- Pequeños cambios en las APIs compatibles con los tests de regresión.
- Uso de mocks.
- Interruptores de características.

¿Y los cambios grandes?

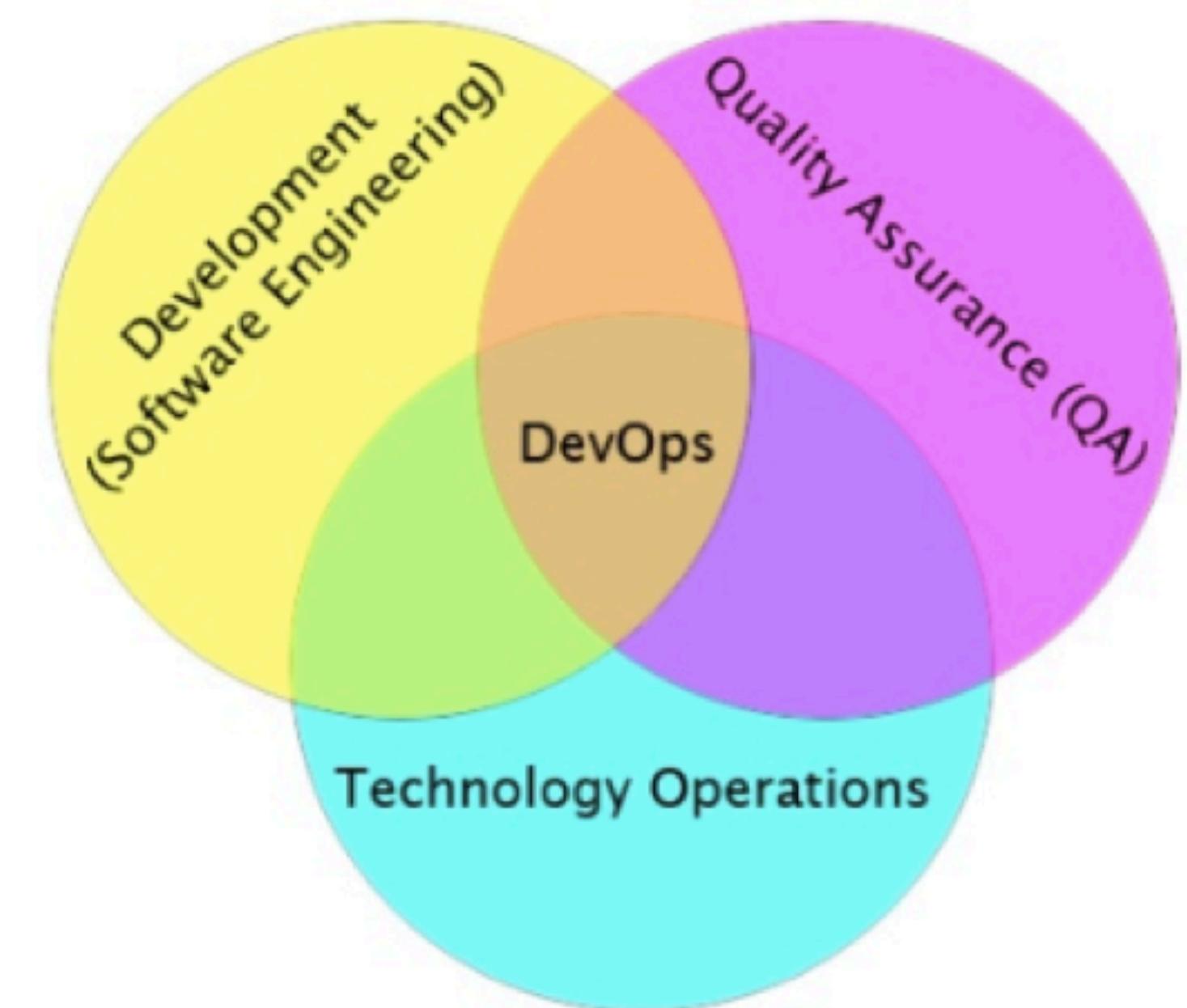
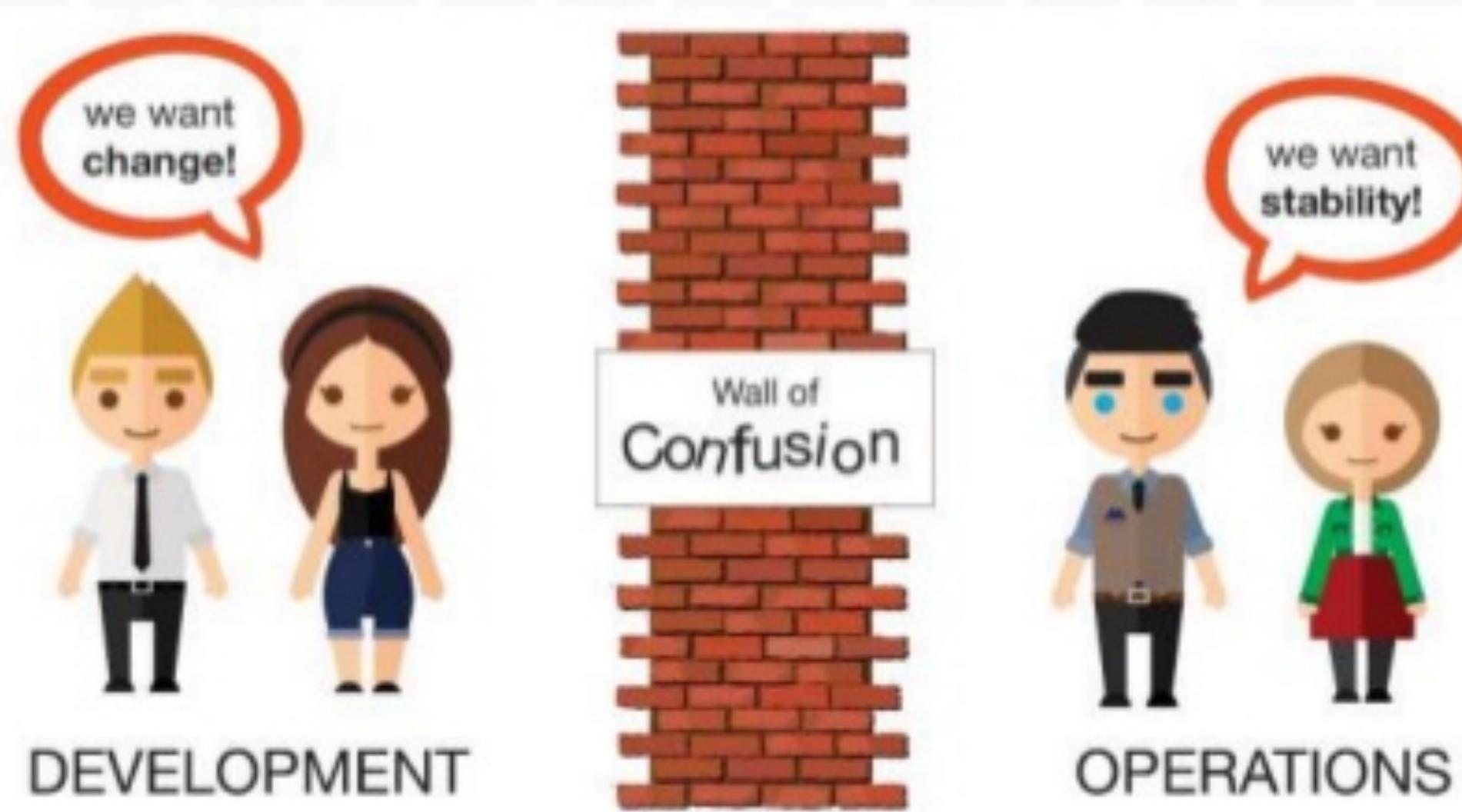
- Codificación y prueba de las pequeñas funcionalidades por separado.
- Buen diseño de código, por ejemplo seleccionar una implementación concreta utilizando interfaces y factorías, pero dejar la estructura lista para introducir futuros cambios.
- Pequeños cambios en las APIs compatibles con los tests de regresión.
- Uso de mocks.
- Interruptores de características.



Canary releases



DevOps



10 mejores prácticas Carto

1. Mantener un repositorio de código
2. Automatizar la compilación
3. Hacer la compilación auto-testable (mediante test automáticos)
4. Todo el mundo realiza commits en la rama principal todos los días
5. Cada commit en la rama principal debe ser compilado
6. Mantener la compilación rápida
7. Testear en un clon del entorno de producción
8. Hacer fácil de obtener los últimos productos compilados
9. Todo el mundo puede ver los resultados de las últimas compilaciones
10. Automatizar el despliegue