



El gran reto del Big Data: la integración continua

Sergio Rodríguez de Guzmán, CTO PUE

**The daily life
of a developer is filled with
monotonous and repetitive
tasks.**

Fortunately,
we live in a pre-artificial
intelligence age, which
means computers are great
at handling boring chores
and they hardly ever
complain about it!



Continuous Integration

- Continuous Integration (CI) is the process of automatically building and testing your software on a regular basis.
- This can be as often as every commit
- Builds run a full suite of unit and integration tests against every commit



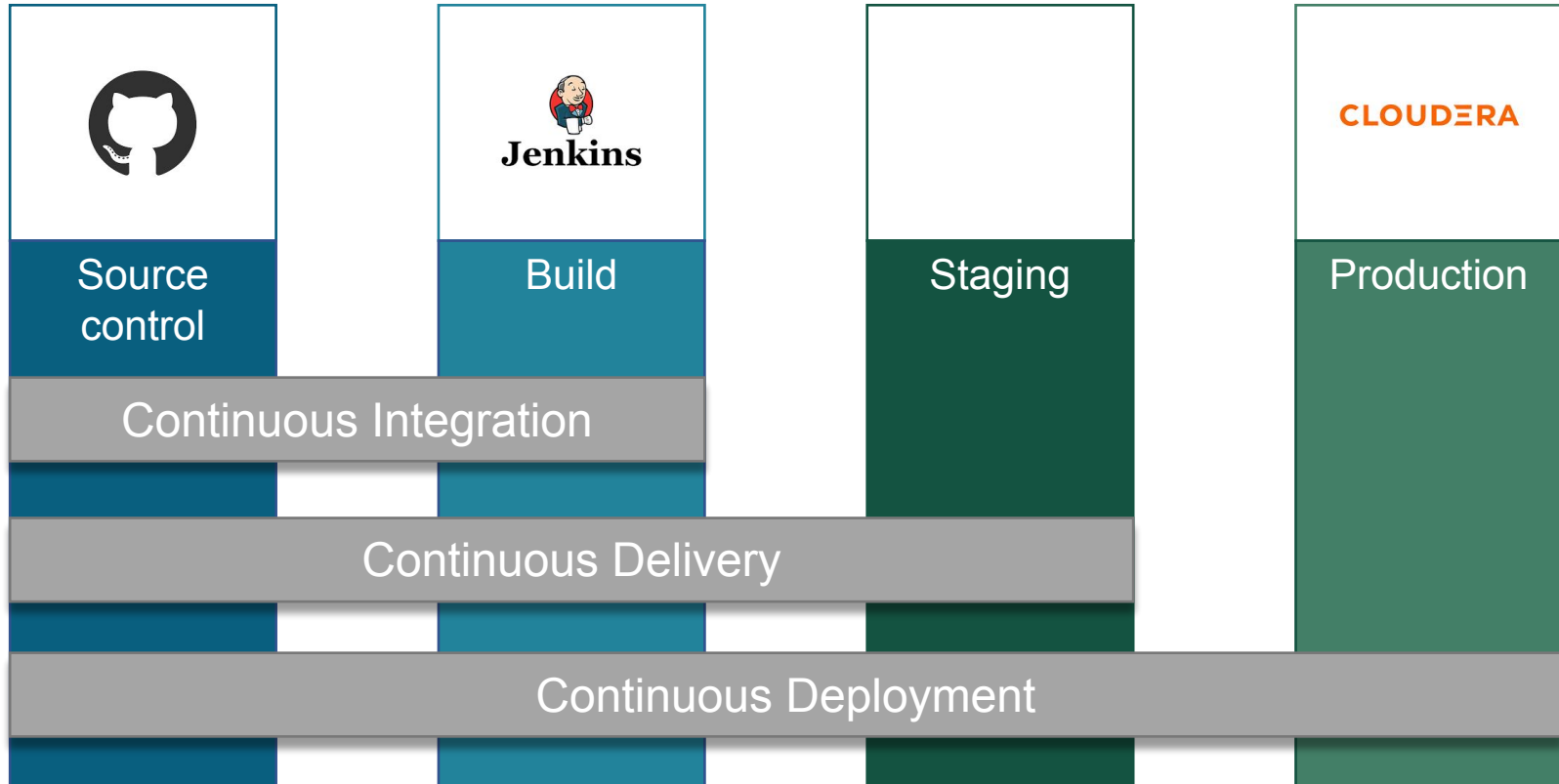
Continuous Delivery

- Continuous Delivery (CD) is the logical next step from continuous integration.
- Continuous Delivery can be thought of as an extension to Continuous Integration which makes us catch defects earlier.
- It represents a philosophy and a commitment to ensuring that your code is always in a release-ready state.



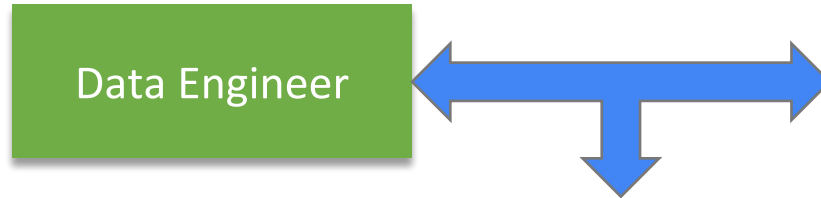
- Continuous Deployment (CD) requires every change to be deployed automatically, without human intervention.
- The ultimate culmination of this process is the actual delivery of features and fixes to the customer as soon as the updates are ready.

[illegible]



Big Data Use Case

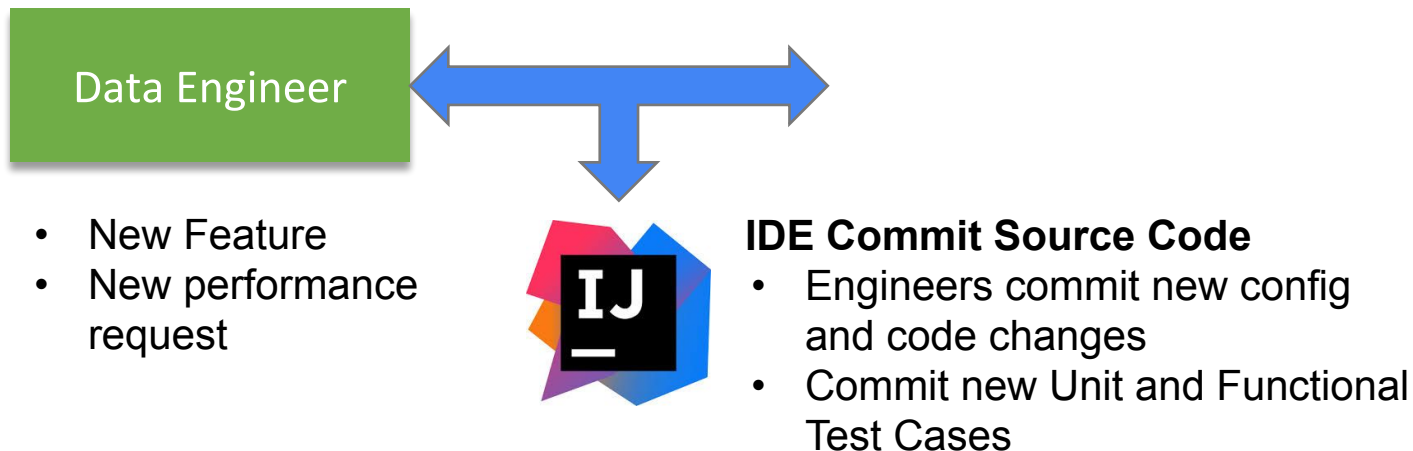
⚡ Jira Service Desk



- New Feature
- New performance request

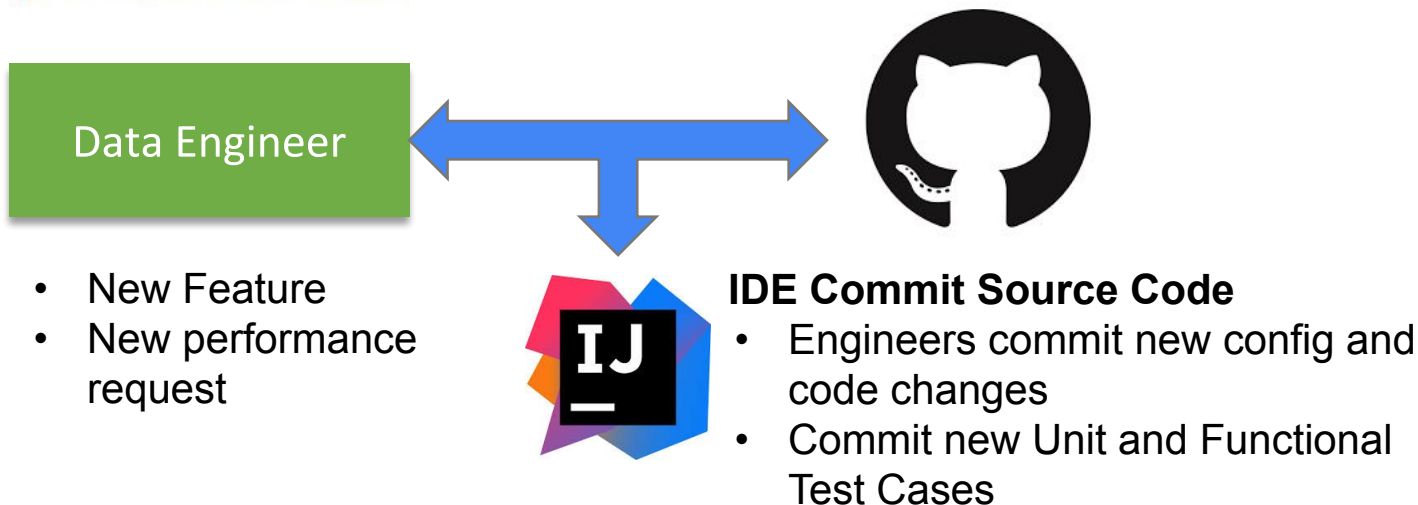
Big Data Use Case

⚡ Jira Service Desk

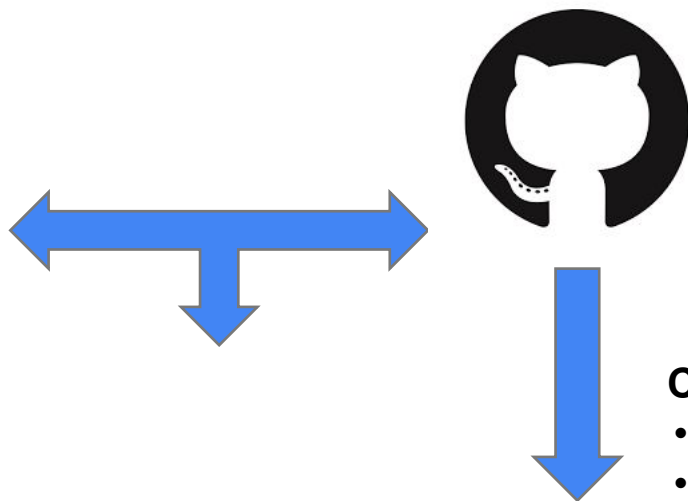


Big Data Use Case

⚡ Jira Service Desk



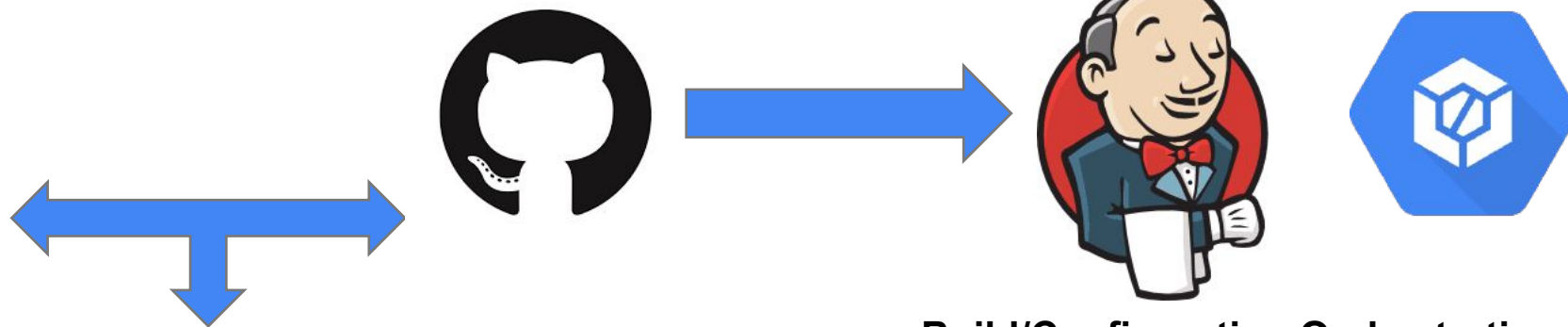
Big Data Use Case



Continuous Notification

- RAG Build Notification
- Test failure for JIRA defects
- Push notifications to JIRA/developers
- Update confluence documentation

Big Data Use Case



Build/Configuration Orchestration

- Code Build and Unit Testing Performed
- ¿Functional and Load tests performed for build release?

Cloud Build

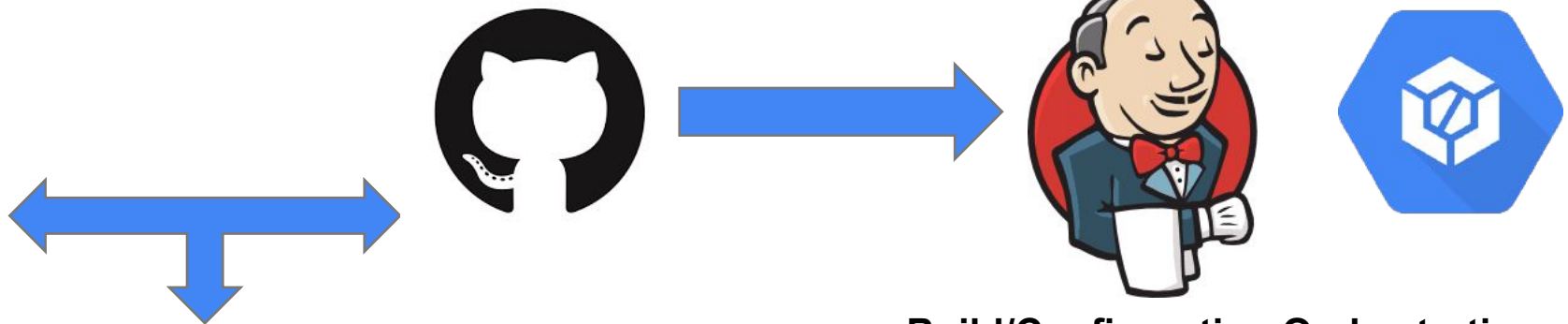


- Docker native compatible
- Vulnerability checks
- Cloud or Local based
- No setup
- YAML configuration pipelines
- GitHub Integration



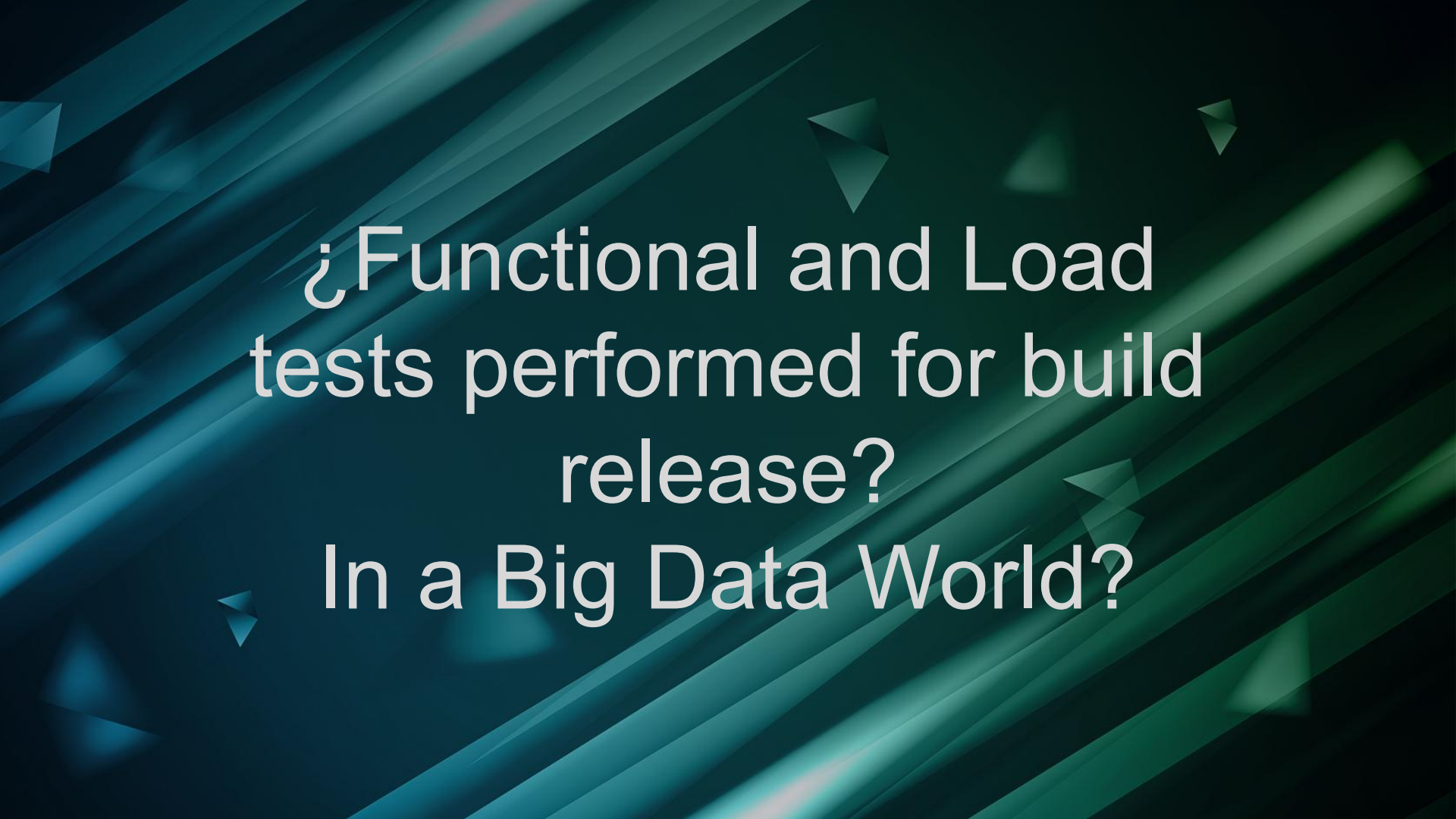
DEMO

Big Data Use Case



Build/Configuration Orchestration

- Code Build and Unit Testing Performed
- ¿Functional and Load tests performed for build release?



¿Functional and Load
tests performed for build
release?

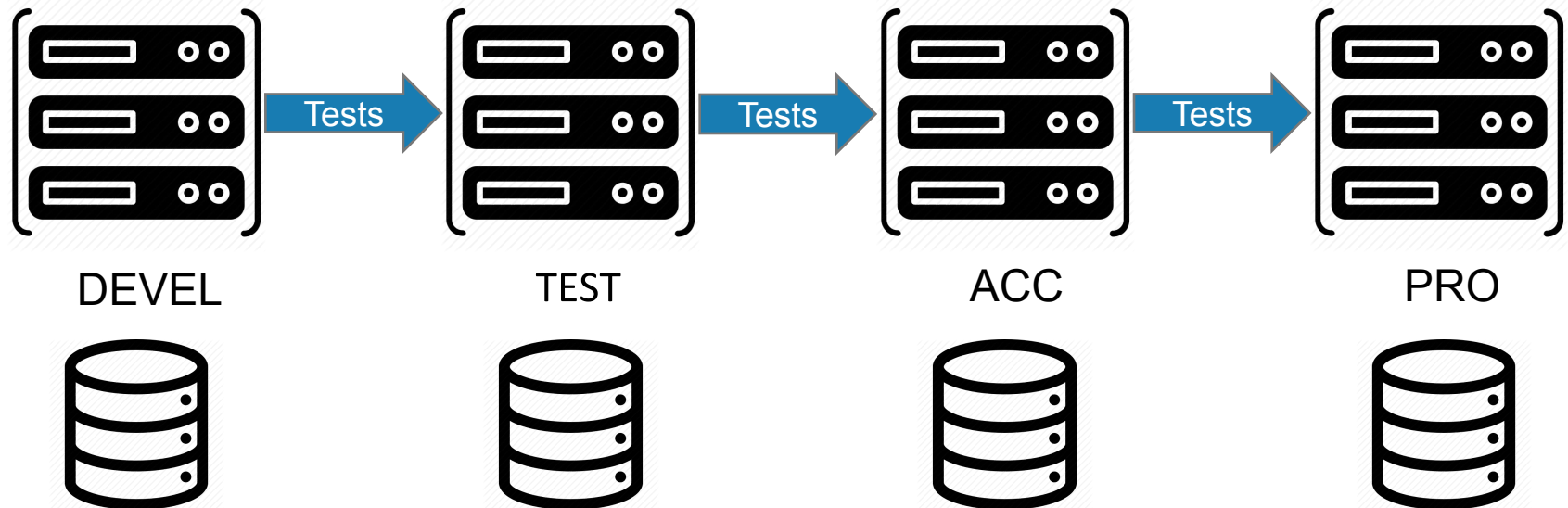
➤ In a Big Data World?

Functional Testing and Load Tests Challenges

- Compute resources
- Storage resources
- Configuration of Services and Apps



Option 1: Multiple Environments



Option 1: Multiple Environments – Pros and Cons

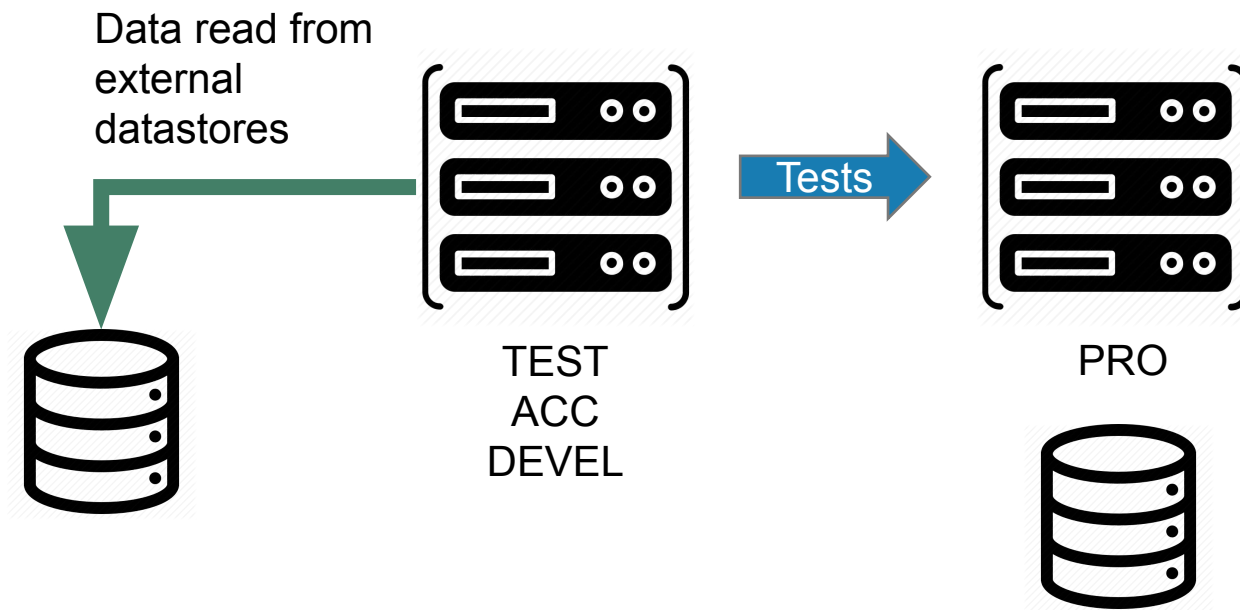
Pros:

- Same sizing as the PRO cluster
- Same configuration
- Same services and security
- Load tests more accurate
- Data sources are the same as in PRO environment ¿?
- Predictable cost
- Flat rate

Cons:

- More maintenance
- More expensive
- Usually 24x7

Option 1: Dynamic Environments



Option 1: Dynamic Environments (Kubernetes)



Hadoop Helm Chart (YARN & MapReduce jobs)

<https://github.com/helm/charts/tree/master/stable/hadoop>

Option 1: Dynamic Environments (Kubernetes)



Apache Spark Helm Chart

<https://github.com/helm/charts/tree/master/stable/spark>

Option 1: Dynamic Environments (Dataproc)



Option 1: Dynamic Environments (Kubernetes) – Pros and Cons

Pros:

- Potentially same sizing as the PRO cluster
- Same services
- Load tests accurate
- Data sources are the same as in PRO environment ¿?
- Low maintenance
- Reduce costs
- Pay as you go

Cons:

- No flat rate
- Need to use external cloud external storage
- Complex initial setup

Option 1: Dynamic Environments (Dataproc) – Pros and Cons

Pros:

- Potentially same sizing as the PRO cluster
- Same services
- Load tests accurate
- Data sources are the same as in PRO environment ¿?
- No maintenance
- Reduce costs
- Pay as you go
- Need to use external cloud external storage

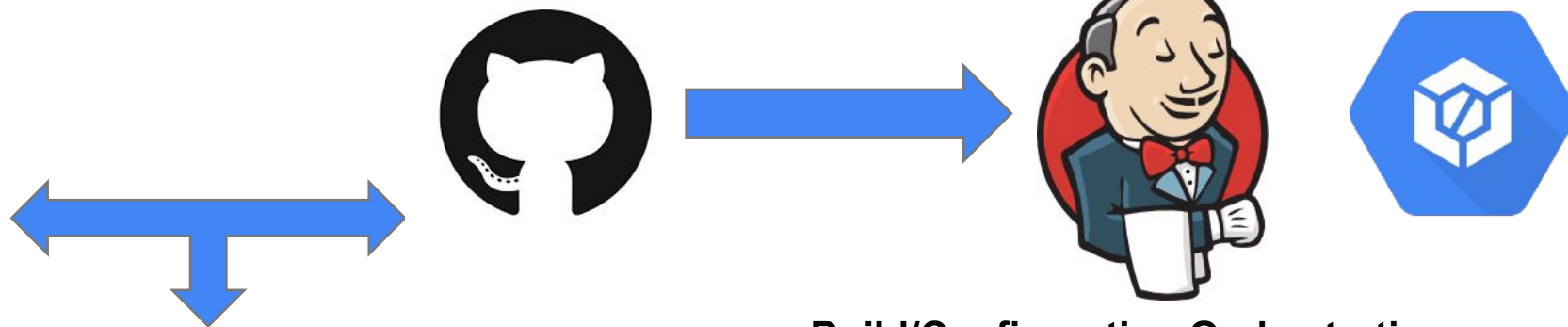
Cons:

- No flat rate
- Need to use external cloud external storage



DEMO

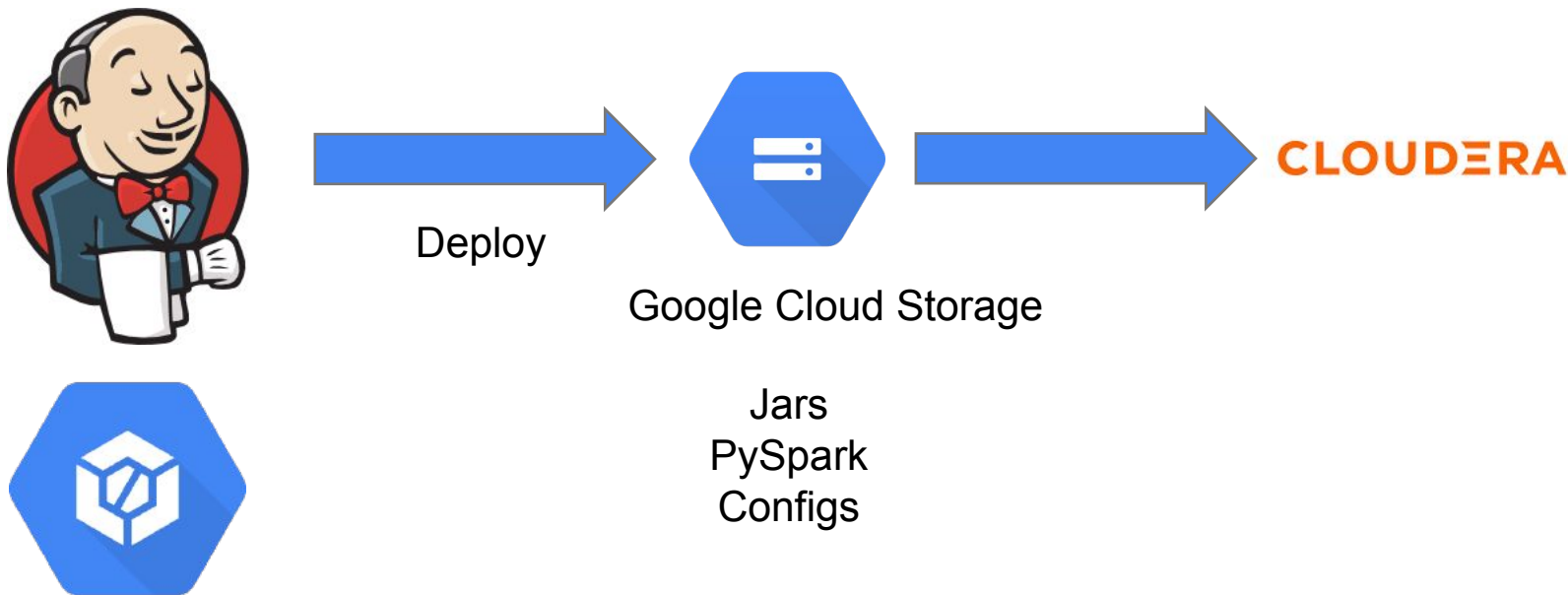
Big Data Use Case



Build/Configuration Orchestration

- Code Build and Unit Testing Performed
- ¿Functional and Load tests performed for build release?

Big Data Use Case – Deploy Option A



Big Data Use Case – Deploy Option B



Deploy

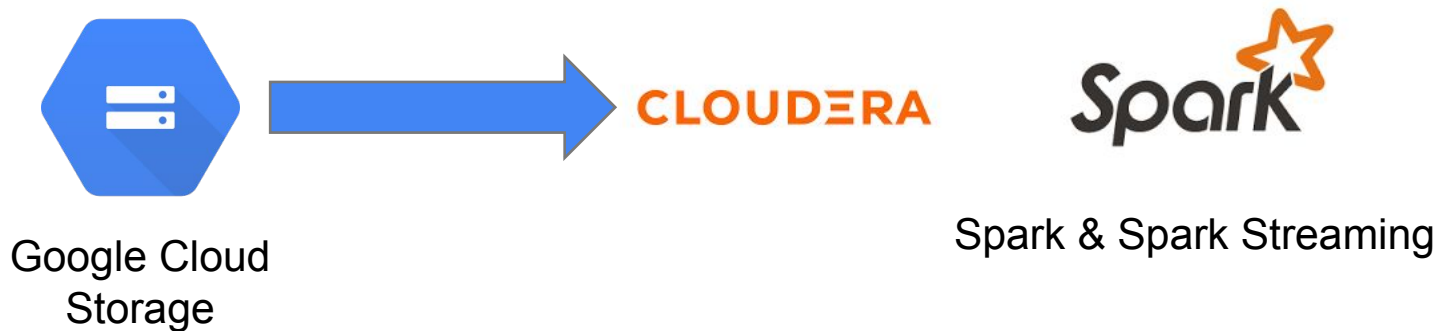


ANSIBLE

CLOUDERA

Jars
PySpark
Configs

Big Data Use Case – Workflow Orchestration



Big Data Use Case – Workflow Orchestration



Google Cloud Storage



CLOUDERA

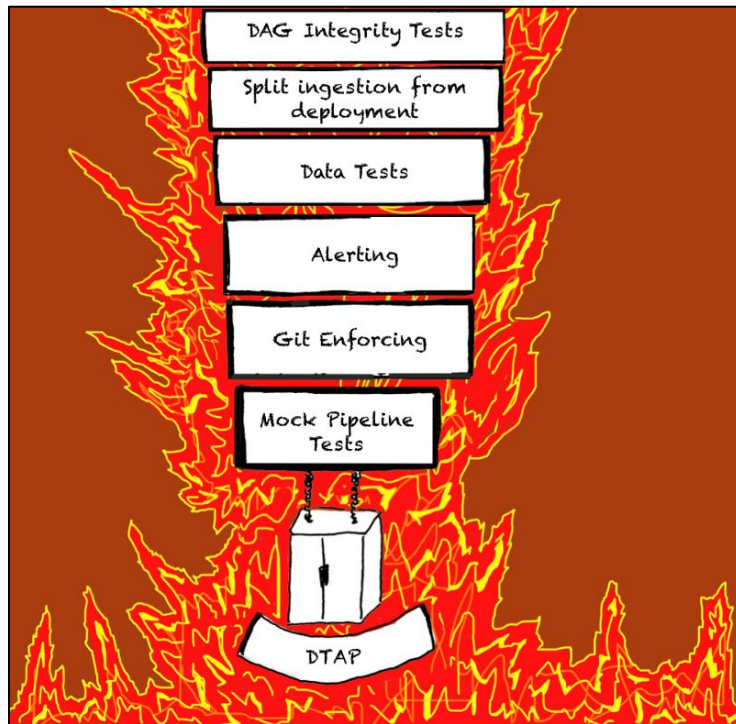


Spark & Spark Streaming



And now?

Big Data Use Case – Data Testing with Airflow



Data Testing Hell – Circle 1

DAG Integrity Tests

Have your CI (Continuous Integration) check if you DAG is an actual DAG.

```
E          airflow.exceptions.AirflowException: Cycle detected in DAG. Faulty task: <Task(BashOperator): templated>

../../../../virtualenv/python3.5.3/lib/python3.5/site-packages/airflow/models.py:2364: AirflowException
===== 1 failed in 0.83 seconds =====
```

Data Testing Hell – Circle 2

Split Ingestion from Deployment

Keep the logic you use to ingest data separate from the logic that deploys your application.

- Create a GIT repository per data source, containing the ETL for the ingestion, and one per project, containing the ETL for that specific project
- Keep all the logic and CI tests belonging to source/project isolated
- Define an interface per logical part

Data Testing Hell – Circle 3

Data Tests

Check if your logic is outputting what you'd expect...

- Are there files available for ingestion?
- Did we get the columns that we expected?
- Are the rows that are in there valid?
- Did the row count of your table only increase?

Data Testing Hell – Circle 4

Alerting

Get slack alerts from your data pipelines when they blow up.

When things go wrong
(and we assume that this will happen),
it is important that we are notified.

Data Testing Hell – Circle 5

Git Enforcing

Always make sure you're running your latest verified code.

Git Enforcing to us means making sure that each day a process resets each DAG to the last verified version (i.e. the code on origin/master).

Data Testing Hell – Circle 6

Mock Pipeline Tests

Create fake data in your CI so you know exactly what to expect when testing your logic.

- Tare two moving parts: the data (and its quality) and your code.
- In order to be able to reliably test your code, it's very important to ensure that your code is the only 'moving part'



Data Testing Hell – Circle 7

DTAP

Split your data into four different environments.

- Development is really small, just to see if it runs
- Test to take a representative sample of your data to do first sanity checks
- Acceptance is a carbon copy of Production, allowing you to test performance and have a Product Owner do checks before releasing to Production

Data Testing Hell – Circle 7

DTAP

DEV

- Quickly run your pipeline on a very small subset of your data
- In our case 0.0025% of all data
- Nothing will make sense, but it's a nice integration test

TST

- Select a subset of your data for data that you know
- Immediately see if something is off
- Still quick to run

ACC

- Carbon copy of production
- You can check if you feel comfortable pushing to PRD
- Give access to a Product Owner for them to check

PRD

- Greenlight procedure for merging from ACC to PRD
- Manual operation
- Great for git blame



¡Gracias!

- ✉ pueacademy@pue.es
- ✉ consulting@pue.es
- ✉ training@pue.es



#PUEAcademyDay19



#PUEihubSessions