Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Ferramenta para automatización de traballos por lotes con Apache Spark

Estudante: Adrián Rodríguez Couto
Dirección: Xoán Carlos Pardo Martínez

A Coruña, febreiro de 2022.

A meus pais, a meu irmán e ós meus amigos, por axudarme durante esta importante etapa.

Agradecementos

Ós meus familiares, en especial meus pais e meu irmán, por todo o apoio e agarimo que sempre me aportaron. Sen eles isto non sería posible.

Ós meus amigos, tanto os que coñecín durante a carreira como os de toda a vida, que o fan todo máis sinxelo e agradábel.

Por último, ó meu titor Xoán que tivo un papel fundamental, por todo o seu esforzo e os consellos que me fixeron posible realizar este traballo .

Resumo

O uso do Big Data, termo que fai referencia á recollida e interpretación de cantidades masivas de datos, está en pleno auxe nos tempos que corren, xa que usando as diferentes tecnoloxías que este término engloba poden chegar a conseguirse beneficios moi claros, axudando aos seus usuarios a sacar conclusións cunha base moito máis sólida e orientarse a decisións efectivas. Dentro deste ámbito do Big Data, entre outras moitas ferramentas, atopámonos co framework open-source de computación paralela Apache Spark, e para facer uso deste co obxectivo de executar traballos por lotes (batch jobs), hai que dispor dun cluster con Spark instalado e enviar os traballos a dito cluster mediante un comando "spark-submit" ou usando unha interface REST.

O proceso de instalación, configuración e uso dun cluster Spark non é sinxelo para usuarios non experimentados debido á cantidade de configuracións que permite e ó elevado número de opcións do comando spark-submit usado para o envío de traballos. Enfocándonos neste último aspecto, o obxectivo deste traballo de fin de grao é elaborar unha ferramenta que poida automatizar e facilitar todo este proceso de execución de traballos por lotes en Apache Spark, permitindo levantar un cluster Spark baixo demanda de forma automatizada, enviar os traballos por lotes que se desexen realizar ao cluster recuperando os logs de execución para a súa interpretación dunha forma sinxela con tal de facilitarlle o uso a usuarios básicos, e por último eliminar o cluster creado.

O cógido fonte da ferramenta desenvolta neste traballo pode atoparse no repositorio git desta ligazón: https://github.com/ADR2211/easySpark

Abstract

The term Big Data refers to the collection and interpretation of extremely large data sets and it's one of the most currently demanded niches in the development and supplement of enterprise software, provoked by the rapid and constant growth of the volumes of information. Big Data technologies provide very clear benefits, assisting users to draw solid conclusions and make effective decisions.

One of the most popular open-source Big Data frameworks for large-scale data analytics is Apache Spark. To run batch jobs on this framework a working Spark cluster is necessary and those jobs are launched using the spark-submit command or using a REST API. Setting up an Apache Spark cluster and use it is not easy for novice users due to the variety of configurations it allows and the multiple options that the spark-submit command supports.

The objective of this Degree Thesis is to develop a tool that automates and facilitates the execution of batch jobs on Spark clusters, providing users with options to deploy and delete a Spark cluster, and allowing inexperienced users to execute batch jobs in a simpler way.

The source code of the tool is available at the following git repository: https://github.com/ADR2211/easySpark

Palabras chave:

- Apache Spark
- Ferramenta de liña de comandos
- Kubernetes
- Cluster
- Vagrant
- Automatización
- Python
- Traballos por lotes

Keywords:

- Apache Spark
- Command line tool
- Kubernetes
- Cluster
- Vagrant
- Automatization
- Python
- Batch jobs

Índice Xeral

Índice de Figuras

Índice de Táboas

Capítulo 1

Introdución

Neste primeiro capítulo móstranse a modo de introdución os principais motivos que levan ao interese de levar a cabo o desenvolvemento deste traballo de fin de grao, os obxectivos que se pretenden cumprir, mencionarase o traballo relacionado con este proxecto tendo en conta ferramentas que proporcionen unha funcionalidade semellante e rematarase explicando de forma sinxela a estrutura deste documento, dando información concreta e resumida acerca do que abarca cada capítulo.

1.1 Motivación

Vivimos nunha época na que se xeran e recollen cantidades masivas de datos xa que o valor destes resulta vital para moitas grandes corporacións. Estes grandes volumes de datos recollidos necesitan ser procesados para converterse en información de valor e o feito de traballar con estes grandes volumes de datos e de ter que procesalos á maior velocidade posible a pesar de poder requirir cálculos intensivos supuxo unha dificultade que o Big Data tivo que resolver.

Estas situacións poderían levarse a cabo cunha arquitectura baseada nun único servidor, o cal non podería paralelizar unha tarefa a alto nivel tendo que asumir dito servidor toda a carga computacional, obtendo como resultado procesamentos de datos de moi longa duración.

Non obstante, o Big Data basease no paradigma da computación distribuída, é dicir, apóiase na existencia dun conxunto de servidores (chamado cluster) que traballan de forma organizada e colaborativa entre eles para resolver un mesmo problema o cal é descomposto nun conxunto de subtarefas que se reparten aos servidores que forman o cluster para que se fagan de forma paralela, o que leva a acadar a realización das tarefas en tempos menores.

Por isto, entre outras diferentes tecnoloxías que tamén seguen o paradigma da computación distribuída explicado, nace Apache Spark, framework de computación en cluster de código aberto moi utilizado para implementar procesos de Big Data e Machine Learning.

Para facer uso do framework Apache Spark, debemos ter un cluster previamente funcional e con Spark configurado, e ditas tarefas de despregar e configurar un cluster Spark para poder facer uso do framework, xunto co paso de enviar traballos por lotes para que se executen contra dito cluster, poden chegar a ser tarefas algo complexas e que poden sobrepasar a un usuario sen experiencia, debido a que dito framework conta con unha gran cantidade de configuracións e moita diversidade en canto a opcións para o envío dos traballos, dificultando que estes poidan usalo.

Ante esta problemática previa, coa idea de facilitar todo o proceso de execución de traballos por lotes contra clusters Spark, chegamos á idea de desenvolver unha ferramenta que automatice a despregadura e o apagado baixo demanda da infraestrutura necesaria xunto coa facilitación da execución dunha das diferentes cargas de traballo que soportan as plataformas Spark, os traballos por lotes, tamén coñecidos como batch jobs.

1.2 Obxectivos

O obxectivo deste traballo de fin de grao é o desenvolvemento de EasySpark, unha ferramenta de liña de comandos para facilitar a execución de traballos por lotes mediante a implementación de subcomandos que permitirán crear un cluster Spark de forma local, executar traballos Spark por lotes no cluster levantado, e por último a opción de eliminar todos os recursos do cluster Spark levantado.

Con esta ferramenta preténdese facerlle ao usuario máis sinxela a execución de traballos no cluster Spark, permitindo a posibilidade de despregar a infraestrutura necesaria de forma automatizada baixo demanda e acomodando o envío de traballos a dita infraestrutura.

1.3 Traballo relacionado

Existen algunhas ferramentas que permiten realizar certas funcións similares ao que aporta este proxecto, non en canto a intentar facilitar a execucion de traballos Spark por lotes, pero si en canto á despregadura e eliminación da infraestrutura necesaria para traballar con Apache Spark, coa diferenza de que só soportan escenarios de uso no cloud (na nube), mentres que este proxecto se centra en escenarios de uso local.

Entre estas ferramentas atópanse:

- **ElastiCluster** [2], unha ferramenta de liña de comandos creada co obxectivo de facilitar a creación, xestión e configuración de clusters de computación aloxados na nube, soportando distintos provedores como Amazon EC2, OpenStack, Microsoft Azure e Google Compute Engine. Naceu como ferramenta para lanzar clusters virtuais HPC (computación de altas prestacións) e na actualidade ademais diso permite levantar clusters Spark/Hadoop, clusters de almacenamento distribuído ou outros tipos que se poidan configurar usando Ansible.

- Distintos servizos en AWS que nos facilitan a despregadura e traballo con clusters Spark, como son **AWS ParallelCluster** [3] e **Amazon EMR** [4]. Por un lado, AWS ParallelCluster é unha ferramenta de administración de clusters de código aberto que facilita a despregadura e administración de clusters HPC en AWS, mentres que Amazon EMR é unha plataforma de cluster administrado que simplifica a execución de frameworks de Big Data diferentes, entre os que se atopa Apache Spark.

- Servizos para traballar co framework Spark co provedor Cloud Azure, como **Azure HDInsight** [5] ou **Azure CycleCloud** [6], unha forma sinxela de administrar cargas de traballo HPC permitindo a despregadura de clusters completos.

- **Dataproc** [7] en caso de traballar con GCP, servizo totalmente xestionado e escalable para executar distintos frameworks entre os que se atopa Apache Spark.

1.4 Estrutura da memoria

Os capítulos que conforman esta memoria de TFG poden verse a continuación, xunto cunha pequena explicación acerca do que abarca cada un:

- ***Conceptos previos:*** Capítulo que brinda información acerca de conceptos teóricos previos necesarios para comprender este TFG.

- ***Tecnoloxías e ferramentas:*** Capítulo no que se recollen as distintas tecnoloxías e ferramentas usadas, xunto cunha explicación acerca destas e a xustificación da súa escolla diante doutras alternativas.

- ***Metodoloxía de desenvolvemento:*** Capítulo dedicado á explicación da metodoloxía que se seguiu para o desenvolvemento da ferramenta EasySpark.

- ***Desenvolvemento do proxecto:*** Capítulo no que se recollen os principais aspectos do deseño e desenvolvemento da ferramenta.
- ***Conclusións e traballo futuro:*** Último capítulo desta memoria no que plasmamos as conclusións sacadas do proxecto, xunto coa proposta de liñas de traballo futuro para a mellora da ferramenta.

1.5 Repositorio do proxecto

O código fonte da ferramenta desenvolvida pode atoparse no repositorio Git da seguinte ligazón: `https://github.com/ADR2211/easySpark`

Capítulo 2

Conceptos previos

Neste capítulo preténdese recoller aqueles conceptos teóricos dos que houbo que documentarse previamente para poder entender e levar a cabo o proxecto abordado, e que son de axuda para contextualizar e comprender o funcionamento da ferramenta desenvolta.

2.1 Big Data

O Big Data é un termo que describe o gran volume de datos que se xeran, almacenan, manexan e son analizados polas empresas na actualidade. Esta análise masiva de datos, que se leva a cabo de forma xeral seguindo un ciclo de xestión da información (fig. 2.1) en 4 pasos diferenciados, leva ás empresas a obter valiosos coñecementos que lles poden axudar ou servir como apoio para a toma de decisións.

Figura 2.1: Fases do ciclo de xestión da información.

Os datos analizados poden clasificarse según a súa estrutura, diferenciando entre:

- Datos estruturados: Información que adoita atoparse en bases de datos, que xa está ordenada e que conta cun formato predeterminado. Arquivos de texto que contan con filas e columnas con títulos que serven de identificadores da información. Exemplos deste tipo de datos son bases de datos relacionais ou follas de cálculo.

- Datos non estruturados: Información que carece de organización e que tampouco conta cun formato predeterminado. Pódese tratar de documentos PDF ou Word, ficheiros multimedia, correos electrónicos ...
- Datos semiestruturados: Mezcla dos dous anteriores tipos de datos. Son datos que non presentan unha infraestrutura completamente organizada e fixa pero que sí presentan unha organización definida mediante etiquetas/metadatos que describen os obxectos e as relacións. Os formatos de intercambio de información XML ou JSON son exemplos deste tipo de datos.

A idea do Big Data é que permite ter acceso a máis información, e canta máis información se teña, maior é o entendemento que se pode acadar e mellor é a toma de decisións realizada. En moitos casos, o proceso de análise dos datos está totalmente automatizado, xa que se conta con ferramentas avanzadas que crean millóns de simulacións para obter os mellores resultados e conclusións posibles, aínda que a necesidade de xestionar tantos datos leva a requisitos de infraestruturas moi estables e potentes normalmente baseadas na computación distribuída (subsección 2.1.1) que teñan capacidade suficiente para soportar todos os procesos necesarios.

O Big Data ten un potencial enorme e é moi importante para o progreso da tecnoloxía, e dado que non para de crecer, xorden diferentes conceptos, tecnoloxías, técnicas e ferramentas para xestionalo, creadas especificamente para levar a cabo estas tarefas de almacenamento e análise de datos masivos. Unha das tecnoloxías que xurdiu debido a isto foi **Apache Spark**, o eixe central deste proxecto. Spark é un framework de computación distribuída que soporta varios métodos de procesamento de datos como son o coñecido como ”Stream processing” ou procesamento de fluxo, que é unha práctica baseada en realizar accións necesarias sobre os datos xa no seu momento de creación, en tempo real, e por outro lado, tamén permite traballar co ”Batch processing” ou procesamento por lotes. Este último método de procesamento será ao se lle dará soporte na ferramenta desenvolvida neste proxecto. Na subsección 2.1.2 explícase con maior detalle en que consiste o procesamento por lotes.

2.1.1 Computación distribuída

A computación distribuída é un modelo de computación en paralelo onde interveñen múltiples computadores que poden ou non estar situados en distintos lugares sobre unha rede distribuída co obxectivo de resolver problemas computacionais. Estes computadores utilizan estándares abertos para levar a cabo unha mesma tarefa ou obxectivo común entre todas elas, dividíndose o problema a resolver en moitas tarefas individuais, sendo cada unha destas divisións resolta por unha ou máis computadoras que se comunican mediante o intercambio de mensaxes.

Este modelo de computación resulta moito máis económico que a opción de ter unha única máquina moito máis potente e que poida resolver soa os problemas, ademais de ofrecer maior escalabilidade e de brindar moita maior dispoñibilidade se o comparamos coa alternativa de ter só unha máquina responsable.

2.1.2 Traballos por lotes

O procesamento por lotes (batch processing) é a execución de traballos que se asignan para ser executados nun ordenador sen máis interacción por parte do usuario durante o proceso e adoita utilizarse para a execución de tarefas de datos repetitivas e de gran volume. Por isto, de xeito simple, os traballos por lotes son programas que se executan sen o control ou supervisión directa do usuario.

Este método de procesamento desempeña un papel fundamental á hora de axudar ás empresas a xestionar grandes cantidades de datos de forma eficiente. A execución de cada traballo por lotes conta cuns mesmos fundamentos, sendo necesarios para cada un destes traballos definir uns parámetros esenciais: quen envía o traballo, que programa se quere executar, a localización das entradas (inputs) a recibir xunto coa localización na que gardar os resultados (outputs) e por último o intre no que se debería executar dito traballo. Normalmente todos os traballos por lotes toman datos de entrada, os procesan e xeran unha saída xunto cuns logs que deben ser almacenados para una análise posterior, tendo entón que xestionarse para unha exitosa execución ese almacenamento temporal e transferencia de ficheiros de entrada/saída dos traballos.

Unha vez un traballo por lotes é enviado, entra nunha fila de espera na cal estará esperando ata que o sistema responsable estea listo para procesalo, podendo ter esta fila moitos outros traballos e establecer a orde de execución mediante o tempo de chegada ou por prioridades. Debido a estas filas de prioridade o procesamento por lotes faise eficiente permitíndolle ás empresas priorizar as tarefas máis urxentes e é por isto que os traballos por lotes normalmente son agregados á fila durante as horas de traballo, pero non se procesan ata as horas libres ou a fin de semana cando os sistemas responsables non están ocupados, aproveitándose da característica de que non necesitan supervisión directa do usuario.

Aínda que este tipo de traballos se executen sen supervisión directa pode suceder que a execución falle por algún motivo, polo que este tipo de procesamento adoita utilizar alertas de administración baseadas en excepcións para notificar ás persoas necesarias en caso de haber problemas, permitindo así que os responsables poidan traballar sen ter que revisar

regularmente o progreso dos traballos por lotes, xa que en caso de que algo non funcione correctamente recibirán unha alerta sobre unha excepción crítica.

2.2 Virtualización de servidores

A virtualización de servidores é unha tecnoloxía que permite crear múltiples ambientes virtuais nun só sistema de hardware físico basándose en técnicas hardware, software ou ambas, de xeito que todos eles comparten os mesmos recursos físicos pero atopándose illados entre si. Isto permite executar múltiples sistemas operativos sobre un mesmo hardware ó mesmo tempo pero simulando que cada un deles se está executando sobre o seu propio hardware dedicado.

Existen diferentes técnicas de virtualización de servidores:

- Emuladores de hardware (Bochs, QEMU).
- Virtualización baseada en hipervisor (VirtualBox, Vmware ESXi).
- Virtualización a nivel de kernel (KVM).
- Virtualización por compartición de kernel (Docker, FreeBSD Jails).

Dependendo do tipo de técnica a utilizar estes ambientes virtuais poden recibir distintas nomenclaturas, destacando no contexto deste proxecto os coñecidos como máquinas virtuais (virtualización basada en hipervisor) e contedores (virtualización por compartición de kernel).

2.2.1 Máquinas virtuais

A virtualización baseada en hipervisor céntrase na utilización de hipervisores, unha capa software que virtualiza o hardware dun servidor físico e xestiona os contornos virtuais, chamados máquinas virtuais neste tipo de virtualización, que se executan nel.

O hipervisor encárgase principalmente de asignar os recursos físicos entre as máquinas virtuais, de proporcionar seguridade e illamento entre as VM e de crear e xestionar a execución das máquinas virtuais. Entre os hipervisores podemos diferenciar 2 tipos:

- Nativo ou Bare Metal: Hipervisor que se executa directamente sobre o hardware do servidor físico, como por exemplo VMware ESXi.
- Hosted: Hipervisor que se executa sobre un sistema operativo xa en funcionamento, chamado SO anfitrión, agregando unha capa intermedia entre o SO sobre o que se executa e as máquinas virtuais (figura 2.2).

Figura 2.2: Representación gráfica da virtualización baseada en hipervisores hosted.

2.2.2 Contedores

A virtualización baseada en contedores, ou tamén chamada ”virtualización a nivel de sistema operativo” ou ”virtualización por compartición de kernel” (fig. 2.3) é un método de virtualización propio de sistemas Linux (na actualidade soporta tamén outros SO) que non require dun hipervisor, e no que os contornos virtuais execútanse coma procesos no espazo de usuario.

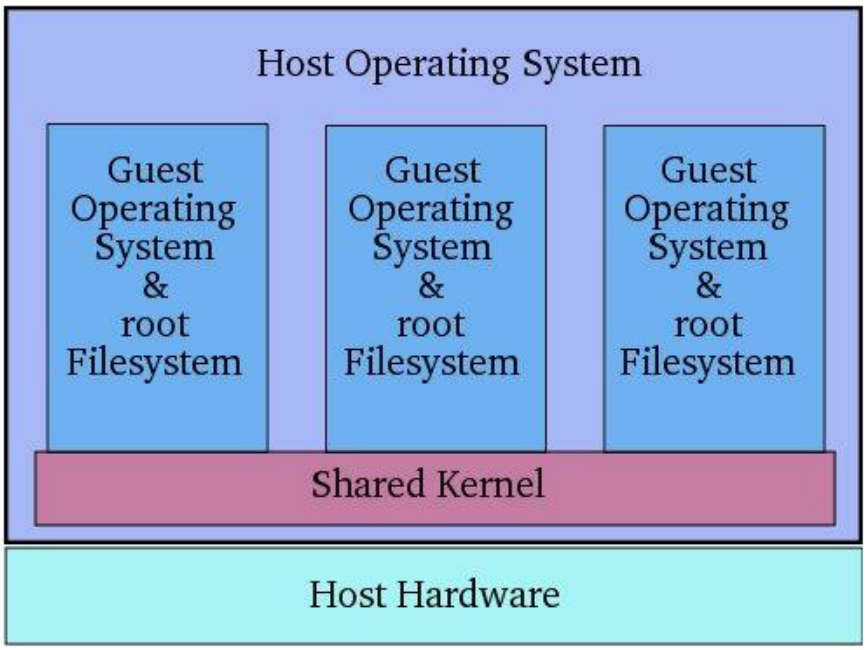

Figura 2.3: Representación gráfica da virtualización baseada en contedores.

Normalmente os sistemas operativos modernos separan a memoria virtual en espazo de kernel e espazo de usuario (fig. 2.4). O espazo de kernel está estrictamente reservado para

executar o kernel privilexiado do sistema operativo, extensions do kernel e a maioría de controladores de dispositivos (comunicación co hardware), mentres que o espazo de usuario é a zona de memoria onde se executan as aplicacións que corremos como usuarios.

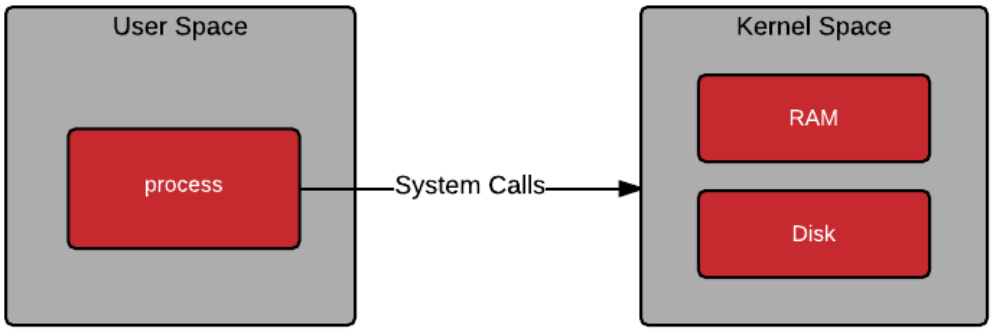

Figura 2.4: Representación espazo de usuario contra espazo de kernel.

Neste tipo de virtualización o sistema operativo proporciona capacidades de virtualización que permiten ter múltiples contornos ou ambientes virtuais executándose sobre o mesmo kernel. Os contornos virtuais execútanse como procesos illados no espazo de usuario, vendo cada un unha versión virtual propia dos recursos dispoñibles, e é por isto que para que se poida usar este tipo de virtualización o kernel do sistema operativo anfitrión debe proporcionar tanto illamento entre procesos como particionamento de recursos e limitación/control de acceso aos mesmos.

Estes contornos virtuais de execución poden recibir diferentes denominacións como zonas, particións, gaiolas ou contedores. Neste proxecto usaranse estes últimos, xa que son os que usa Docker para proporcionar contornos illados para a execución de aplicacións.

2.3 IaC: Infraestrutura como código

O paradigma IaC (Infrastructure as Code) ten como obxectivo realizar a xestión da infraestrutura informática a través da escritura de código que permita definir, aprovisionar e/ou xestionar infraestrutura de forma automatizada, proporcionando unha alternativa ó método tradicional de xestión de sistemas que se veu aplicando ata os tempos actuais a través de ferramentas GUI (interface gráfica de usuario) que precisan interacción e a execución de comandos por CLI (interface de liña de comandos).

A utilización do paradigma IaC trae consigo múltiples vantaxes:

- Permite automatizar por completo todo o proceso de aprovisionamento e despregadura de infraestruturas, sen necesidade da interacción dos administradores, o que se traduce en despregaduras máis rápidas e fiables.
- Os ficheiros de código que representan a infraestrutura poden ser versionados, ó igual que o código dunha aplicación, permitindo así contar cun histórico de cambios que pode ser de utilidade de cara á resolución de problemas.
- Debido a que o despregue/xestión se representa no código, este pode ser validado mediante revisións de código e tests automatizados.
- Reutilización de código.
- Permite representar o estado da infraestrutura en arquivos de código, de forma que calquera con coñecementos os pode ler e interpretar.

Na actualidade existen múltiples ferramentas IaC (figura 2.5), con diferentes características e funcionalidades, as cales é habitual ter que combinar para conseguir o obxectivo de despregar a infraestrutura. Entre ditas ferramentas, pódense diferenciar según o seu obxectivo principal, separando entre ferramentas para o aprovisionamento da infraestrutura (Terraform, Pulumi), ferramentas para a xestión da configuración (Chef, Puppet, Ansible, CFEngine,SaltStack), ferramentas para a xestión de imaxes/modelos (Docker, Vagrant) e ferramentas para a orquestración da infraestrutura (Kubernetes), aínda que estes 4 conxuntos mencionados non son necesariamente disxuntos.

Figura 2.5: Exemplos de ferramentas IaC.

Dentro do gran número de ferramentas IaC que existen, as que se utilizan neste proxecto e se explicarán en detalle no capítulo de Tecnoloxías e Ferramentas (3) son Vagrant, Docker e Kubernetes, figura 2.6.

Figura 2.6: Ferramentas IaC utilizadas no proxecto.

Capítulo 3

Tecnoloxías e ferramentas

Neste terceiro capítulo introdúcense as tecnoloxías e as ferramentas que se empregaron para a realización do proxecto, explicando para que serve cada unha delas.

3.1 Apache Spark

Apache Spark[8] é un motor de procesamento de datos distribuídos de propósito xeral que é moi utilizado nun gran número de circustancias. Pertence, tal e como o seu nome indica, á Apache Software Foundation, o que garante a licenza de código aberto e ademais da linguaxe de programacion Scala na que está implementado, ofrece soporte para outras linguaxes como Python, Java e R. Algunhas das tarefas máis frecuentes asociadas a Spark inclúen traballos Extract, Transform and Load (ETL) e SQL por lotes sobre grandes conxuntos de datos, procesamento de datos en tempo real de sensores ou IoT por exemplo, e tarefas de Machine Learning.

Spark basease nun compoñente principal ou core, e sobre el existen distintos compoñentes que estenden o seu uso, destacando os que mostra a figura 3.1:

- **Spark Core**: É o núcleo do framework, xa que é a base ou conxunto de librerías onde se apoian o resto de módulos.
- **Spark SQL**: Permite realizar consultas de tipo SQL sobre unha base de datos relacional, xa sexa mediante Apache Hive, ODBC, JDBC ou outras. É un módulo para o procesamento de datos estruturados e semi-estruturados que permite transformar e realizar operacións sobre os dataframes ou RDD, nomes que reciben as abstraccións usadas en Spark para representar os conxuntos de datos distribuidos.
- **Spark Streaming**: Este compoñente estende o uso de Spark para permitir realizar procesamento de datos en tempo real a través dun sistema de agrupamento de pequenos lotes.

- **MLib**: Librería moi completa que contén numerosos algoritmos de Machine Learning, tanto de agrupamento (clustering), clasificación, regresión... facilitando así a implementación distribuída de solucións baseadas na Intelixencia Artificial.
- **GraphX**: Este módulo permite realizar cálculos paralelos sobre grafos.

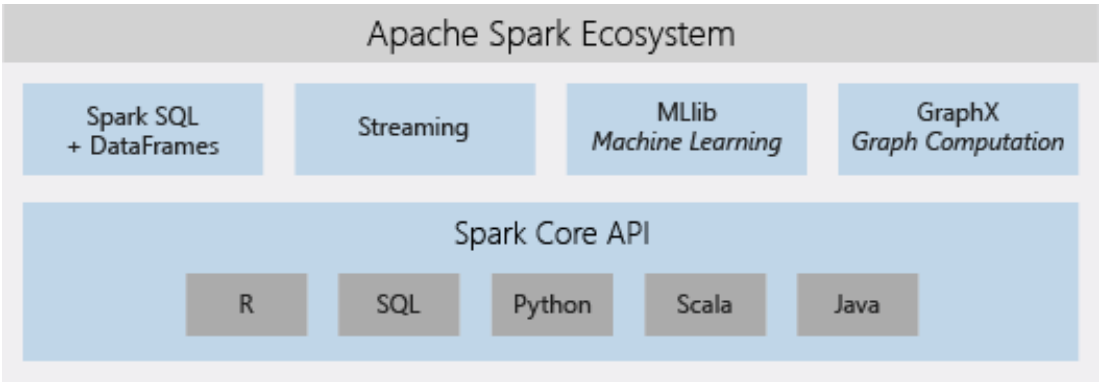

Figura 3.1: Compoñentes de Apache Spark

Apache Spark traballa con diferentes abstraccións básicas que representan coleccións de datos distribuídos como poden ser os Datasets, os DataFrames ou as táboas SQL pero de entre as distintas abstraccións que soporta, a principal son os RDD[9][10], o tipo de dato máis básico que ten Apache Spark. Os RDD son unha colección de datos capaz de operar en paralelo e tolerante a fallos, xa que se atopan distribuídos polos diferentes nós do cluster Spark e pódense rexenerar as distintas particións dun RDD volvendo a aplicar as transformacións que o calculan a partir doutros RDD previamente almacenados. Son inmutables, polo que en caso de que se queiran facer operacións sobre eles terase obrigatoriamente que xerar un novo e adoitan crearse desde arquivos HDFS, o sistema de ficheiros de Apache Hadoop, un contorno de traballo para aplicacións distribuídas co que pode integrarse Apache Spark. Desta forma, un RDD conta con múltiples particións de datos almacenadas en distintos nós do cluster (Spark pode almacenar datos redundantes para ter a capacidade de recuperarse ante erros), e sobre estes RDD están soportadas 2 tipos de operacións:

- ***Transformacións***: Son operacións que permiten transformar os datos, funcións que collen un RDD como entrada e producen un ou varios RDD á saída, debido a que non poden cambiar a entrada ao ter a propiedade de seren estruturas inmutables. Execútanse en modo *lazy* (o cálculo atrásase ata a execución dunha operación de tipo "Acción"), polo que son chamadas non bloqueantes para unha aplicación Spark. A orde na que estas operacións de transformación foron executadas almacénase, para que en caso de erro poidan recalcularse as particións ou RDD concretos, simplemente volvendo a calcular as transformacións que se lles aplicaron.

- ***Accións***: Operacións en Spark que devolven o resultado final dos cálculos/transformacións sobre o RDD, son chamadas bloqueantes (a diferenza das transformacións) converténdose así en desencadeadoras da execución das transformacións pendentes ata o momento. Xeralmente permiten obter información do RDD sobre o que se executan como pode ser solicitar o número de elementos ou simplemente facer que os datos sexan almacenados en disco.

Respecto ao modelo de execución de Spark, está baseado no paradigma MapReduce, do que aproveita os beneficios de ser escalable, distribuído e tolerante a fallos pero conseguindo ser máis eficiente e sinxelo de utilizar. Porén, este framework conta tamén coas *operacións Map, Reduce e Shuffle and Sort* propias do paradigma MapReduce. Spark vai construíndo un grafo acíclico dirixido ou DAG con todas as transformacións que se realizan sobre un RDD ata que a execución dunha acción desencadee a execución do grafo. Desta maneira, Spark utiliza os enlaces no grafo para representar as operacións *Shuffle and Sort* tratando de agrupar todo o que sexa posible as tranformacións nos nós (un nó do grafo pode conter varias operacións). Con isto, cada nó do grafo representaría un traballo MapReduce, e para cada nó do grafo e cada partición do RDD defínese unha tarefa que debe ser executada por un nó executor.

3.1.1 Arquitectura de Spark

En canto á arquitectura básica, o framework Apache Spark está deseñado principalmente para ser instalado e configurado sobre un cluster de computadores, e non sobre un único equipo.

Trátase dun framework que admite moitas configuracións diferentes, o que dificulta moito a súa instalación e preparación por parte de usuarios sen experiencia, e un exemplo disto pódese ver na variedade de xestores de cluster (cluster managers) cos que pode traballar. O cluster sobre o que se configura Spark pode ter como xestor do cluster diferentes opcións como *Hadoop YARN*, *Apache Mesos*, *Kubernetes* ou o xestor de cluster propio do framework *Spark standalone cluster manager*.

Para executar unha aplicación Spark, o framework obtén os recursos necesarios e planifica a execución de tarefas mediante o xestor de recursos do cluster. Existen diferentes métodos posíbeis para o envío de aplicacións ó cluster, non obstante, neste TFG centrámonos unicamente nunha das opcións, a ferramenta de liña de comandos ”spark-submit”.

Os procesos que compoñen a execución dunha aplicación Spark (fig. 3.2) consisten nun driver e un número variable de executors que corre como un conxunto independente de procesos no cluster, tendo cada aplicación os seus propios executors que quedan en funcionamento

ata o seu remate, conseguindo así o beneficio de que as aplicacións estean illadas entre sí, tanto pola parte dos drivers (cada un ten as súas tarefas) como pola parte dos executor. Isto implica que os datos non poden ser compartidos entre distintas aplicacións Spark a non ser que sexan escritos nun almacenamento externo.

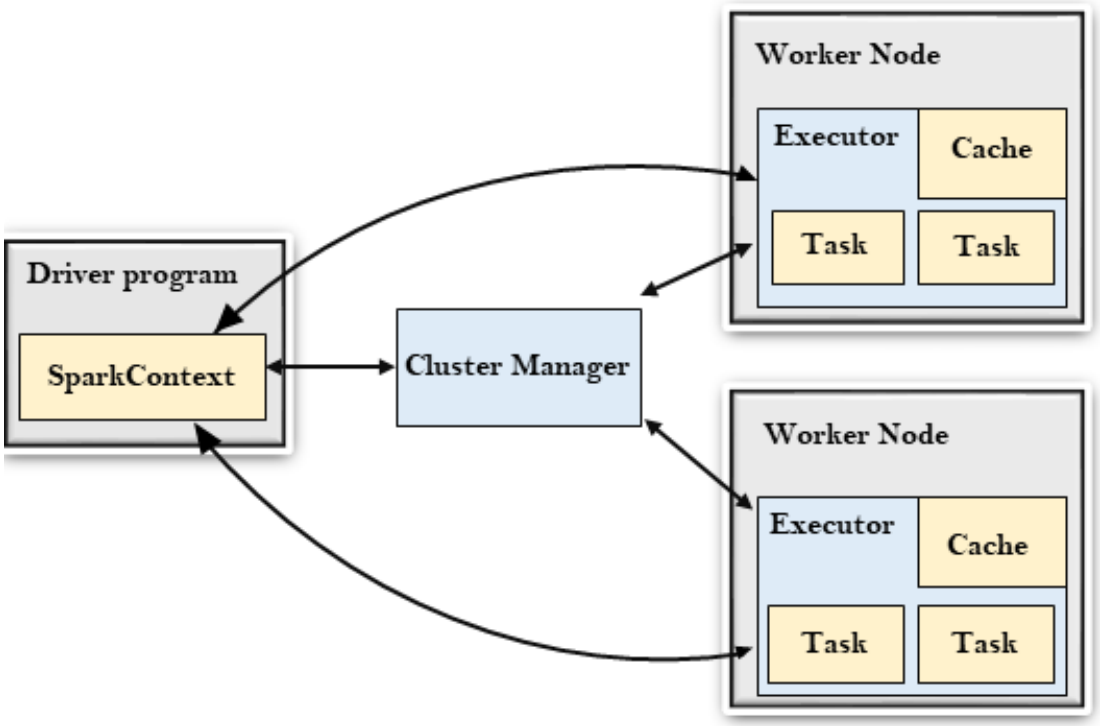

Figura 3.2: Arquitectura básica de Apache Spark

En referencia ao driver, proceso que corre a función main da aplicación a executar e que crea o SparkContext, podemos elexir entre 2 modos de despregadura diferentes, que influirán sobre onde será aloxado dito driver asociado á aplicación (sempre relación 1 a 1 entre dito proceso e unha aplicación en execución):

- ***Cluster mode***: É a maneira máis común de utilizar Spark. O usuario sobe un ficheiro .JAR precompilado ao cluster e o cluster manager lanza o driver nun nó worker do cluster.
- ***Client mode***: Igual que o Cluster Mode, pero a diferenza está en que o driver permanece no equipo cliente que inicia a aplicación. Desta forma, o cliente é o responsable de manter o driver e o cluster manager de manter os executors.

As *aplicacións Spark* tradúcense nun conxunto independente de procesos no cluster, coordinadas polo obxecto SparkContext do programa principal que representa a conexión ao

cluster Spark. Para executar a aplicación sobre un cluster, o SparkContext permítenos conectarnos aos 4 tipos de cluster managers nomeados anteriormente, encargándose de asignar recursos entre aplicacións. Unha vez conectado, Spark consegue procesos "executors" nos nós do cluster (o número a conseguir destes é configurable), procesos que serven para executar cálculos e almacenar datos para a aplicación que se está executando. Tras isto, envíaselles o código da aplicación (definido no JAR ou nos arquivos Python que lle pasamos ao SparkContext) ós executors, e finalmente o propio SparkContext envíalles as tarefas planificadas a cada un dos executors para que as executen, terminando por informar o resultado dos cálculos ao driver que leva o control da aplicación que executamos.

3.2 Kubernetes

Kubernetes [11] é unha plataforma de orquestración de contedores de código aberto deseñada por Google e doada á Cloud Native Computing Foundation para a automatización da despregadura, escalabilidade e manexo de aplicacións en contedores. Está programada na linguaxe de programación concorrente Go e converteuse no estándar de facto para a orquestración de contedores co apoio de compañías clave coma Google, Microsoft ou Intel.

Cando despregamos Kubernetes obtemos un cluster formado por distintos compoñentes que podemos ver na figura 3.3, a cal representa unha arquitectura simple de Kubernetes, xa que se trata dunha arquitectura escalable e por exemplo en contornos de producción, o plano de control corre normalmente en varias computadoras e o cluster ten varios nós, conseguindo así tolerancia a fallos e alta dispoñibilidade. Podemos diferenciar 2 grupos de compoñentes, os *compoñentes do plano de control* e os *compoñentes dos nós*.

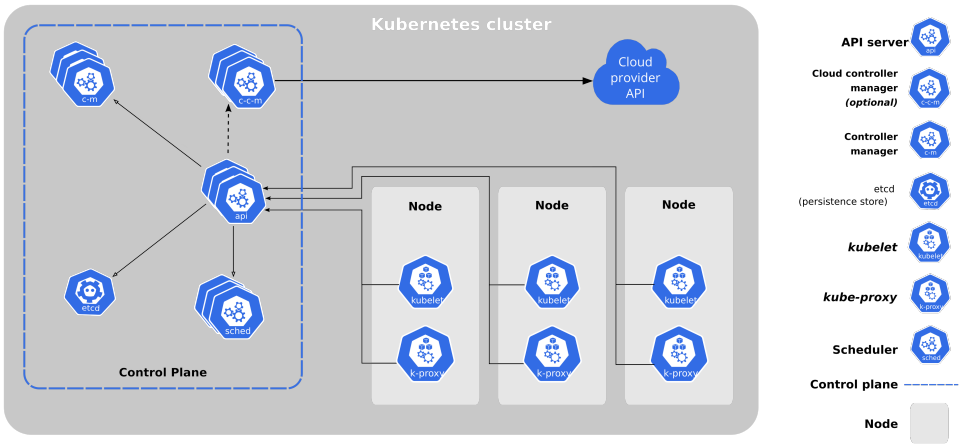

Figura 3.3: Compoñentes que conforman un cluster Kubernetes

3.2.1 Compoñentes do plano de control

O plano de control controla os nós worker e os Pods, unidade de computación despregable máis pequena que se pode crear e xestionar en Kubernetes, e que representan un conxunto de contedores en execución nun cluster. Os compoñentes que o forman toman decisións globais acerca do cluster e ademais detectan e responden aos distintos eventos que ocorren nel, como pode ser iniciar un novo Pod cando non se cumpre o requisito do número de réplicas dunha despregadura. Os compoñentes son:

- *kube-apiserver*: Compoñente que expón a API kubernetes, sendo o front end do plano de control. Recibe as peticións e actualiza o estado en etcd.
- *etcd*: Almacén de datos persistente, consistente e distribuído de clave-valor utilizado para almacenar a información do cluster Kubernetes.
- *kube-scheduler*: Compoñente que revisa e se encarga de asignar onde executar os Pods que non teñen ningún nó asignado a través de diferentes factores.
- *kube-controller-manager*: Executa os controladores de Kubernetes, bucles de control que revisan o estado do cluster e executan ou solicitan os cambios que sexan necesarios para alcanzar o estado desexado.
- *cloud-controller-manager*: Ao igual que o anterior compoñente executa controladores, pero neste caso son controladores que interactúan cos provedores da nube. Este compoñente permite eliminar as dependencias das funcionalidades específicas de cada provedor cloud.

3.2.2 Compoñentes de nó

Os compoñentes de nó corren en cada nó do cluster, mantendo os Pods en funcionamento e proporcionando o contorno de execución de Kubernetes.

- *kubelet*: Axente que corre en cada un dos nós asegurándose que os contedores estean correndo nun Pod. Este axente colle un conxunto de especificacións de Pod (PodSpecs) que se proporcionan a través de distintos mecanismos e asegúrase de que os contedores que o forman estean funcionando e en bo estado.
- *kube-proxy*: Proxy de rede que corre en cada un dos nós do cluster, implementando parte do concepto de servizo de Kubernetes, é dicir, a forma na que se expón unha aplicación executada en Pods como un servizo de rede. Permite abstraer o servizo de Kubernetes mantendo as regras de rede no anfitrión e facendo reenvío de conexións.

- *container-runtime*: O software responsable de executar os contedores. Kubernetes soporta varios como por exemplo Docker, containerd ou CRI-O.

3.3 Minikube

Minikube [12] é unha ferramenta de código aberto que permite executar Kubernetes de forma local facilmente. Está soportado en diferentes sistemas operativos como macOS, Linux e Windows e utilízase para despregar un cluster Kubernetes dentro dun contorno virtual sobre o equipo no que se utiliza.

O contorno virtual sobre o que desprega o cluster pode tratarse de máquinas virtuais, contedores ou sobre servidor físico (bare-metal), dependendo do driver (software de virtualización) que se decida utilizar na execución da ferramenta. Algún dos softwares de virtualización son Docker, VirtualBox, VWare fusion ou HyperV, e a escolla destes marcará o tipo de contorno virtual sobre o que se fará a despregadura.

3.4 Vagrant

Vagrant [13] é unha ferramenta de CLI dispoñible para Windows, MacOS e GNU/Linux que permite xerar contornos de desenvolvemento reproducibles e portables dunha forma moi sinxela a través da creación e configuración de máquinas virtuais a partir de simples ficheiros de configuración coñecidos como Vagrantfiles. É un software libre desenvolvido pola empresa HashiCorp que orixinalmente só permitía usar o provedor VirtualBox, pero actualmente é capaz de traballar cunha gran multitude de provedores como VMware, Amazon EC2, LXC e DigitalOcean, aínda que no ámbito do noso proxecto só sexa utilizada co provedor VirtualBox. Debido ao grande uso que ten na actualidade, esta ferramenta conta cunha gran comunidade a cal está moi activa.

O feito de que permita crear e configurar as máquinas virtuais a través de ficheiros de configuración de texto aporta grandes vantaxes, permitindo que estes ficheiros poidan ser compartidos (e versionados con ferramentas de control de versións como ”Git”) entre equipos de varios desenvolvedores e ter a garantía así de que a través destes todos e cada un dos integrantes traballan co mesmo contorno de desenvolvemento, ademais de solucionar problemas de compatibilidades de software con algúns sistemas operativos.

A arquitectura de Vagrant pódese observar na figura 3.4. Os principais compoñentes son:

- ***Vagrantfile***: Documento de texto que segue a sintaxe da linguaxe Ruby que ten como función principal describir o tipo de máquina/s a levantar no proxecto, como configurala e como aprovisionala. Hai un único Vagrantfile por proxecto, e o contorno de desenvolvemento configurado no Vagrantfile levántase executando o comando "vagrant up".

- ***Boxes***: As boxes son o formato de paquete para os contornos Vagrant. Poden ser utilizadas en calquera plataforma que soporte Vagrant para levantar un contorno de desenvolvemento idéntico.

- ***Provisioners***: Permiten instalar automaticamente o software, alterar as configuracións e máis cousas como parte do proceso do "vagrant up". Son moi útiles debido a que normalmente as boxes utilizadas como base non están construídas especificamente para o caso de uso a tratar, e a pesar de que a instalación de software tamén se poida facer manualmente conectándonos ó contorno levantado, estes provisioners permiten automatizar o proceso facéndoo repetible.

- ***Providers***: O concepto de provider fai referencia ao motor de virtualización utilizado para a despregadura do contorno e a súa API. Vagrant soporta unha gran cantidade de motores de virtualización, por defecto ven con soporte para VirtualBox, Hyper-V e Docker pero a través do sistema de plugins de Vagrant consegue dar soporte a moitos outros como VMware.

Vagrant usouse neste proxecto a través dun módulo Python, que actúa como wrapper da interface de liña de comandos de Vagrant, para levar a cabo a despregadura en local do cluster Apache Spark en modo standalone.

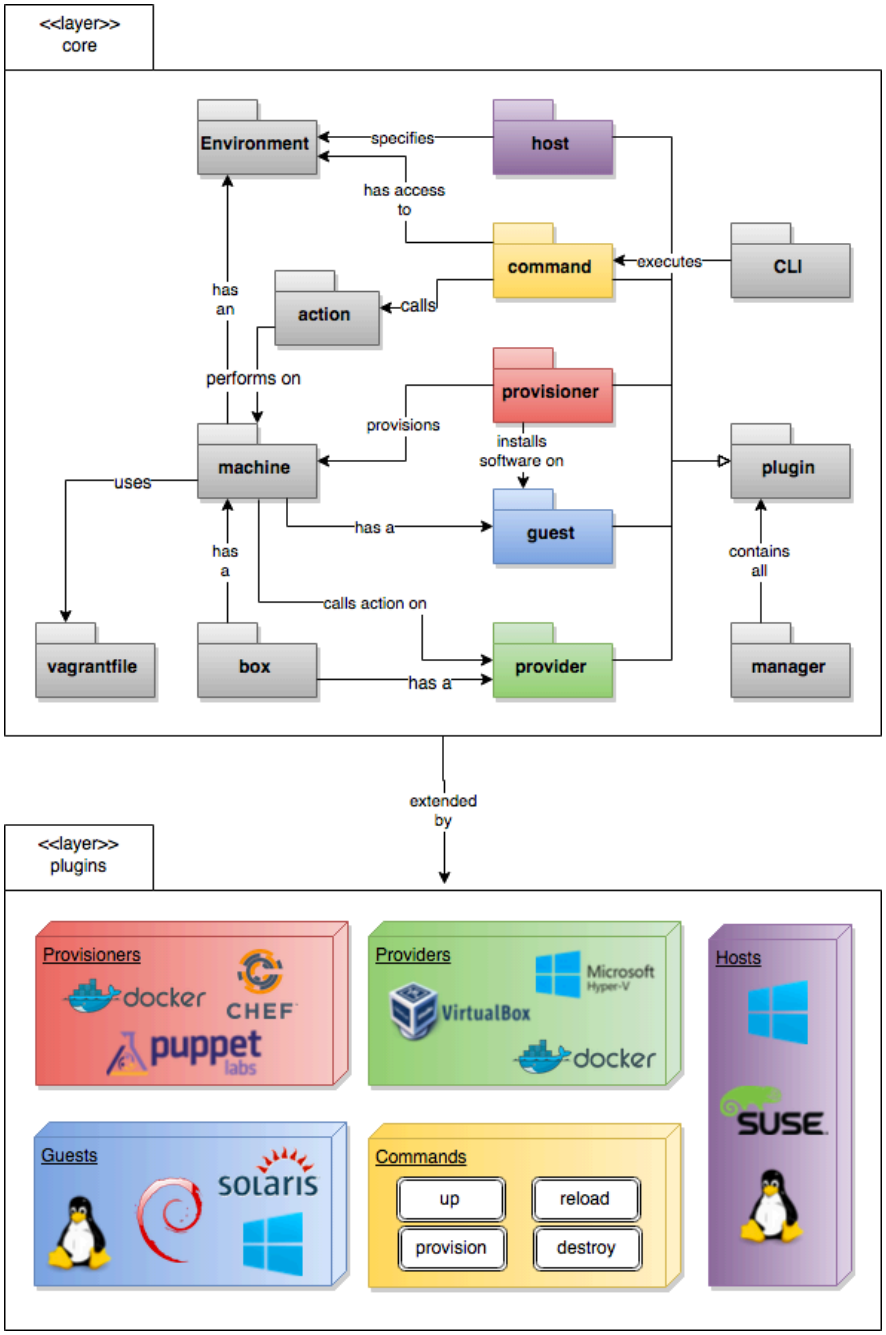

Figura 3.4: Arquitectura básica de Vagrant.

3.5 VirtualBox

VirtualBox [14] é un software de virtualización para arquitecturas x86/amd64 que permite crear máquinas virtuais dentro das limitacións de hardware do host anfitrión. Dito software

ofrece dúas alternativas diferentes para usalo dende a interface de usuario ou mediante liña de comandos (VBoxManage) e mediante este é posible instalar distintos sistemas operativos como sistemas operativos invitados dentro do sistema operativo "anfitrión", permitíndonos traballar con contornos con diferente ou igual sistema operativo ao do propio equipo anfitrión.

Este software está dispoñible para hosts Windows, Linux, MacOS e Solaris e soporta unha ampla lista de sistemas operativos invitados.

3.6 Ferramentas de desenvolvemento

3.6.1 Python

Python [15] é unha linguaxe de programación de código aberto que pon énfase na lexibilidade do código e o aumento da produtividade, e é multiparadigma, xa que soporta a orientación a obxectos, a programación imperativa e en parte tamén a programación funcional.

Foi a linguaxe elexida para desenvolver o proxecto por dúas razóns: Por un lado, é unha linguaxe moi utilizada para automatización, debido a que é sinxela, con scripts, conta cun bo soporte por parte da comunidade e ten unha gran cantidade de ferramentas e módulos para moitos tipos de traballos. Por outro lado, tamén influíu que é unha linguaxe coa que apenas traballei anteriormente e me gustaría aprender.

3.6.2 Visual Studio Code

Visual Studio Code [16] é un editor de código fonte lixeiro pero potente desenvolto por Microsoft dispoñible para Windows, macOS e Linux. Ven con soporte incorporado para JavaScript, TypeScript e Node.js pero conta cun rico ecosistema de extensións para outras moitas linguaxes como Python ou C++, ademais de que inclúe soporte para depuración, control integrado con Git, resaltado de sintaxe e finalización intelixente de código.
Foi o editor escollido debido a que xa traballara anteriormente con el.

3.6.3 Git

Git é un software de control de versións gratuíto e de código aberto, deseñado pensando na eficiencia e para manexar tanto proxectos pequenos como grandes con rapidez. Foi utilizado durante o desenvolvemento deste proxecto para levar o control das funcionalidades que se

ían implementando de forma sinxela, podendo consultar sempre as implementacións previas e incluso volver a estados anteriores no caso de ser necesario.

Capítulo 4

Metodoloxía de desenvolvemento

UNHA metodoloxía de desenvolvemento serve para estruturar, planificar e controlar o proceso de desenvolvemento dun proxecto a través de diferentes etapas coa finalidade de axudarnos a conseguir os obxectivos, sendo así unha decisión moi importante en canto ao desenvolvemento dun produto software a elección dunha metodoloxía axeitada. Neste caso, tendo claro que queriamos seguir unha metodoloxía áxil, de entre as distintas opcións a metodoloxía seleccionada finalmente para utilizar como exemplo a seguir no desenvolvemento deste TFG foi *Scrum*, inda que aplicando certas modificacións e mantendo certas distancias para adaptalo ao contexto no que nos atopamos, no que todos os roles que propón a metodoloxía serán asumidos por 2 únicas persoas, o alumno e o titor.

Neste capítulo explicaremos os principios nos que se basea Scrum [17], os diferentes eventos que a conforman e as relaxacións que aplicamos sobre a metodoloxía para poder tela como base a seguir pero adaptándoa ao proxecto.

4.1 Introdución

Scrum foi desenvolto inicialmente por Ken Schwaber e Jeff Sutherland, conta cunha guía oficial soportada por estes [1], e é un marco de traballo que se utiliza dentro de equipos que manexan proxectos complexos para facilitar a xeración de valor a través de solucións adaptables a problemas complexos. Co seu uso, aplícanse de maneira regular un conxunto de boas prácticas para traballar colaborativamente en equipo co obxectivo de obter o mellor resultado posible dentro dun proxecto.

Mediante esta metodoloxía áxil un proxecto execútase en ciclos temporais e cortos de duración fixa (iteracións de normalmente 2 semanas) coñecidos como *Sprints*, e ao final de cada un destes ciclos tense que proporcionar un resultado completo, un incremento do produto

final que sexa susceptible de ser entregado ao cliente cando o solicite.

4.2 Fundamentos da metodoloxía

Scrum basease no empirismo e no Pensamento Lean, e emprega un enfoque iterativo e incremental para optimizar a previsibilidade e controlar o risco. Ademais, involucra a grupos de persoas que de forma conxunta contan con todas as habilidades e experiencia para facer o traballo e compartir ou adquirir tales habilidades conforme sexa necesario.

Esta metodoloxía áxil combina catro eventos formais para a inspección e adaptación dentro dun sprint, e ditos eventos funcionan porque implementan as tres características ou piares máis importantes de Scrum (fig. 4.1):

- ***Transparencia***: No método Scrum todos os implicados teñen coñecemento do que ocorre no proxecto e de como ocorre, conseguindo así un entendemento común do proxecto.
- ***Inspección***: Frecuentemente os membros que conforman o equipo Scrum inspeccionan o progreso para detectar posibles problemas como mecanismo para saber que o traballo flúe e que o equipo funciona de maneira autoorganizada.

  A inspección permite o seguinte piar de Scrum, a adaptación, e unha inspección sen adaptación non aporta valor.
- ***Adaptación***: Se algún aspecto do proceso se desvía dos límites aceptables ou se o produto resultante non é aceptable, o proceso que se está a aplicar ou os materiais que se producen deben axustarse, xa que esa é a clave para o éxito en proxectos complexos onde os requisitos son cambiantes ou pouco definidos. Dito axuste debe realizarse á maior brevidade posible para minimizar a desviación adicional e espérase que un equipo de Scrum se adapte no momento no que aprende algo novo por medio da inspección.

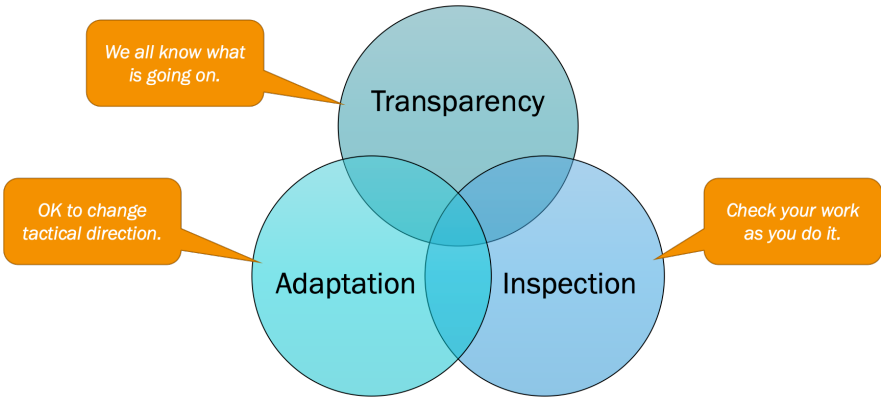

Figura 4.1: Piares empíricos de Scrum [1]

4.3 Equipo

Os equipos Scrum son multifuncionais, o que quere dicir que os membros teñen todas as habilidades necesarias para crear valor en cada sprint, e tamén son autoxestionados, xa que internamente deciden quen fai que, cando e como. É unha unidade cohesionada de profesionais enfocada a un obxectivo común, o produto, e dentro do equipo non existen sub-equipos nin xerarquías.

Non adoitan ser equipos moi grandes, xa que en xeral os equipos máis pequenos comunícanse mellor e son máis produtivos, e dentro destos equipos a metodoloxía Scrum define tres responsabilidades específicas:

- ***Product Owner***: É o responsable de maximizar o valor do produto resultante do traballo do equipo Scrum, inda que a forma na que isto se fai pode variar amplamente entre organizacións, equipos Scrum e individuos.

  Ademais, tamén é o responsable da xestión do *Product Backlog*, que inclúe, tal e como se menciona na guía oficial [1]: o desenvolvemento e comunicación explícita do obxectivo do produto, a creación e comunicación clara de elementos de traballo pendente do produto, o pedido de elementos de traballo pendente do produto e o feito de asegurarse de que o traballo pendente do produto sexa transparente, visible e comprendido.

  Dito Product Owner pode encargarse de todo o traballo anterior mencionado ou delegalo a outros, pero en calquera caso a responsabilidade segue sendo deste.

- ***Equipo de desenvolvemento***: Son as persoas que se comprometen a crear calquera aspecto dun incremento útil (funcional) en cada sprint. É un equipo autoorganizado, coa finalidade de ter unha responsabilidade compartida en caso de non chegar a completar

todas as tarefas dun sprint, sendo os propios membros os que elixen as tarefas das que se encarga cada un. As habilidades específicas que necesitan estes a menudo son amplas e varían conforme o dominio do traballo e ditos equipos adoitan estar formados por entre 3 e 9 profesionais que se responsabilizan da creación dun plan para o sprint, coñecido como *Sprint Backlog*, e de adaptar o seu plan cada día cara o obxectivo do sprint.

- ***Scrum Master***: Responsable de que as técnicas Scrum sexan comprendidas e aplicadas, e consígueo axudando a todos a comprender a teoría e a práctica de Scrum, tanto dentro do equipo como en toda a organización, xa que serve tanto a desenvolvedores, como ao Product Owner, e incluso á organización. Por este motivo, require que sexa un perfil experimentado nesta metodoloxía.

  Son os responsables da efectividade do equipo Scrum pero actúan como axudantes do equipo, non como directores, e no caso de que se decida que non funciona ben, pode ser substituído.

4.4 Artefactos

Os artefactos de Scrum (fig. 4.2) representan traballo ou valor e están deseñados para maximizar a transparencia da información clave co obxectivo de que todos teñan unha mesma visión do que está ocorrendo no proxecto.

Segundo a propia guía de Scrum [1], esta metodoloxía conta con tres artefactos, e cada artefacto contén un *commitment* para garantir que proporciona información que mellora a transparencia e o enfoque co que se pode medir o progreso:

- ***Rexistro de produto (Product Backlog)***: Listado de tarefas que engloba todo un proxecto. Calquera cousa que debamos facer debe estar neste product backlog e cun tempo estimado polo equipo de desenvolvemento.

  O Product Owner ten como responsabilidade ordenar dito product backlog, xa que este membro é o que se atopa en constante comunicación co cliente para asegurarse de que as prioridades están ben definidas, e é a súa responsabilidade que as tarefas que estean máis arriba sexan as de máis prioridade.

  O equipo de desenvolvemento elixe as tarefas do product backlog no *Sprint Planning* para xerar tanto o *Sprint Backlog* como o *Sprint Goal*.

- ***Rexistro de sprint (Sprint Backlog)***: Grupo de tarefas do *Product Backlog* que o equipo de desenvolvemento elixiu no *Sprint Planning* xunto co plan para poder levalas a cabo.

Debe ser coñecido por todo o equipo para asegurarse de que o foco do traballo debe estar sobre estas tarefas.

Conforma unha imaxe moi visible e en tempo real do traballo que os desenvolvedores planean realizar durante o sprint para lograr o obxectivo do sprint. Compre mencionar que o *Sprint planning* (explicado na sección 4.5) non pode cambiarse durante o sprint, só o plan que se decidiu para poder desenvolver as tarefas.

- ***Incremento***: Un incremento é un paso cara o obxectivo do produto. Cada un destes incrementos é aditivo a todos os incrementos anteriores e verificado a fondo, asegurando que todos os incrementos funcionen xuntos. Estes deben proporcionar valor, polo que o incremento debe ser utilizable.

  Pode haber varios incrementos dentro dun sprint e a suma de incrementos preséntase na *Sprint Review* (explicado na sección 4.5). Ademais, un incremento pode ser entregado ás partes interesadas antes do final do sprint.

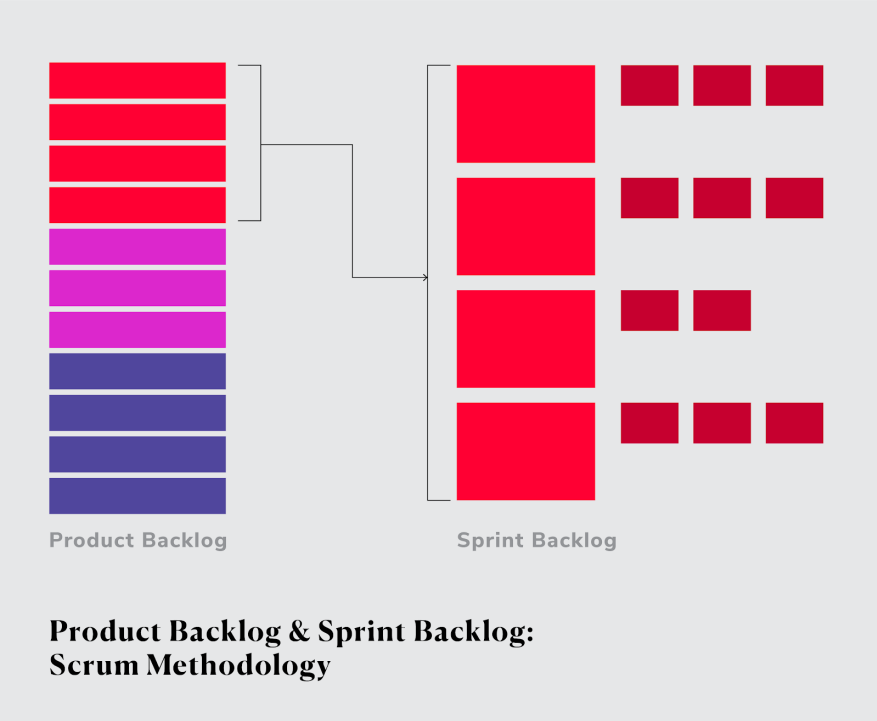

Figura 4.2: Artefactos da metodoloxía Scrum.

4.5 Eventos

Scrum enfócase na xestión das persoas, definindo unha serie de eventos cuxa finalidade principal, inda que non única, é a coordinación do equipo Scrum, e usando estes eventos para permitir tanto a transparencia necesaria como para crear regularidade e minimizar a necesidade de reunións non definidas. Os diferentes eventos definidos dentro do marco Scrum son:

- ***Sprint***: É o corazón de Scrum e todo o traballo necesario para alcanzar o obxectivo do produto (Product Goal), incluíndo os eventos Scrum restantes (Sprint Planning, Daily Scrums, Sprint Review e Sprint Retrospective) ocorren dentro dos sprints (fig. 4.3).
  Son eventos de duración fixa, 1 mes como máximo, para crear unha consistencia e un novo sprint comeza inmediatamente despois da conclusión do sprint anterior. Non é posible atrasar o prazo de entrega dun sprint, polo que en caso de que houbera atrasos, a funcionalidade que non se realizara móvese para o seguinte sprint.
  Estes sprints permiten a previsibilidade ao garantir a inspección e adaptación do progreso cara o obxectivo do produto, polo que hai que ter coidado cos sprint duradeiros xa que poden provocar que se perda realimentación valiosa do cliente e poñer así en perigo o proxecto.

- ***Planificación do sprint (Sprint Planning)***: Este plan é creado de forma colectiva por todo o equipo Scrum e é o evento que inicia o sprint establecendo o traballo que se realizará neste. Nel elíxense as entradas do Product Backlog que se levarán a cabo no sprint, descompoñéndoas en tarefas técnicas con esforzo estimado, concluíndo este evento coa creación do Sprint Backlog para o sprint actual.

- ***Scrum diario (Daily Scrum)***: Evento que ten o propósito de inspeccionar o progreso cara o obxectivo do sprint e adaptar o Sprint Backlog conforme sexa necesario. É unha reunión diaria que debería durar como máximo 15 minutos e na que interveñen os desenvolvedores co obxectivo de mellorar a comunicación, identificar impedimentos e promover unha rápida toma de decisións.

- ***Revisión do sprint (Sprint Review)***: Evento que ten como propósito revisar o resultado do sprint e determinar futuras adaptacións. Nel, o equipo presenta o resultado do traballo ás partes interesadas e revísase o logrado durante o sprint e os cambios que houbo, e en base a esta información, discútese o progreso cara o obxetivo do produto e deciden entre todos que facer a continuación .

- ***Retrospectiva do sprint (Sprint Retrospective)***: É o último evento dun sprint en Scrum e ten como propósito planificar formas de aumentar a calidade e a eficacia. Faise

unha valoración de como se implementou a metodoloxía Scrum nese sprint, analízase que foi ben durante o sprint, que problemas houbo e como foron ou non foron resoltos co obxectivo de conseguir unha lista de melloras que pasarán a ser tidas en conta xa no seguinte sprint, que comeza ao rematar este evento.

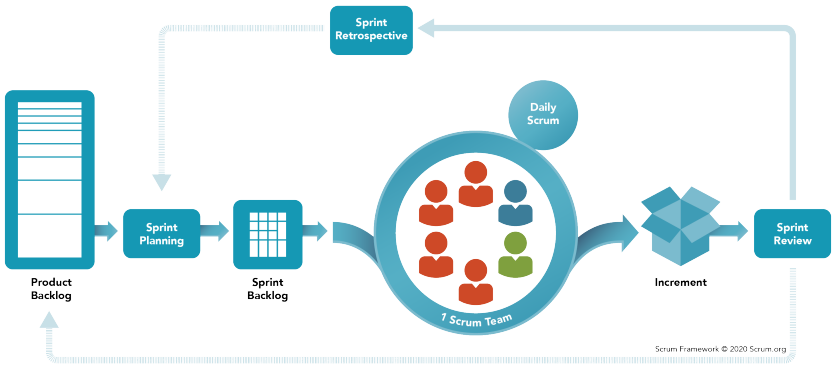

Figura 4.3: Diagrama dun sprint Scrum cos eventos e artefactos que engloba.

4.6 Aplicación de Scrum no TFG

A metodoloxía Scrum utilízase habitualmente para levar a cabo proxectos complexos a través dun equipo Scrum, ten unhas pautas moi ben definidas e foi a elexida como metodoloxía de desenvolvemento durante este proxecto, pero para o contexto do TFG no que nos atopamos, tivemos que aplicar unhas certas adaptacións ad-hoc.

Por unha parte, Scrum está orientado a xestionar equipos de persoas, tendo equipos de desenvolvedores de entre tres e nove persoas, pero no caso no que nos atopamos, o equipo de desenvolvedores está formado por unha única persoa, que será o alumno que presenta o proxecto. Ademais, esa mesma persoa desempeña o rol de Scrum Master, asegurándose de que se seguen os pasos da metodoloxía Scrum. Por outro lado, a outra persoa participante no proxecto, o titor, desenvolveu os roles de persoa interesada ou cliente, e ademais de Product Owner, intentando maximizar o valor do proxecto.

En canto ao tempo de duración dos sprints en Scrum, adoita ser fixo, pero no caso no que nos atopamos non foi así debido á indispoñibilidade durante certos intervalos de tempo de ambos os 2 membros implicados no proxecto, fixándose como duración de sprint o tempo entre as reunións de seguimento entre o alumno e o director. Ademais, non se levou a cabo o Daily Scrum estrito demandado pola metodoloxía, subsituíndose pola comunicación dos

avances e problemas segundo se foran producindo.

Capítulo 5

Desenvolvemento do proxecto

Neste capítulo detállase o proceso de desenvolvemento que se levou a cabo para implementar a ferramenta resultante do proxecto.

O capítulo divídese en diferentes seccións, onde a maioría destas representan unha iteración de desenvolvemento (sprint dentro da metodoloxía Scrum). Dentro de cada unha destas seccións referentes aos sprints, séguese unha base común en canto ao abordado durante a iteración, e debido a isto cada unha destas seccións divídese nas seguintes subseccións:

- Análise: Estudo dos principais obxectivos a alcanzar na iteración actual.
- Toma de decisións e desenvolvemento: Toma de decisións sobre as alternativas para o desenvolvemento dos casos de uso e implementación das funcionalidades abordadas na iteración.

Ademais das seccións relativas ás distintas iteracións, tamén agregamos 4 seccións adicionais nas que comentaremos: a preparación previa ao comezo do proxecto, a linguaxe usada e o tipo de aplicación desenvolta xunto cos motivos disto, a planificación inicial do proxecto coas suas estimacións de custo e tempo, os resultados finais de tempo e custo a modo de comparativa co planificado na primeira fase e unha sección referente ás probas realizadas.

5.1 Estudo previo e decisións iniciais

Antes de comezar co desenvolvemento da ferramenta de liña de comandos resultante deste proxecto, levouse a cabo unha procura de información exhaustiva acerca do dominio sobre o que se traballaba, recollendo información sobre en que consistía Apache Spark, para que se utilizaba e realizando varias probas de execución neste tipo de clusters despregando os escenarios necesarios para levalas a cabo, co obxectivo de conseguir certo coñecemento e soltura

co framework no que se basea o proxecto.

Tras isto, por unha parte, pasouse á elección da linguaxe de desenvolvemento escollendo finalmente Python, unha linguaxe moi popular e moi fácil de aprender, portable (pode funcionar en calquera plataforma como Windows/ MacOS / Linux), que conta cunha gran comunidade de apoio e que ademais se adapta moi ben para a elaboración de ferramentas de liña de comandos entre outras cousas debido ao amplo conxunto de bibliotecas e módulos cos que conta para moitas funcionalidades distintas. Ademais dos outros factores mencionados, tamén tiveron o seu peso as gañas de aprender esta linguaxe.

Por outro lado, en canto ao tipo de aplicación, co obxectivo de automatizar todos os pasos necesarios para a execución de traballos por lotes en Spark, decidímonos por elaborar unha aplicación de liña de comandos (interface de usuario baseada en texto) fácil de usar en lugar de elaborar unha aplicación con GUI, xa que consideramos que se adaptaba mellor ao que buscamos debido a que nos brinda maior simplicidade, necesita menos recursos en comparación con outras interfaces e que adoita ser unha opción moi común en traballos que buscan a automatización de tarefas, podendo usarse en scripts por exemplo.

5.2 Planificación

As fases principais de traballo deste proxecto foron as seguintes:

1. Procura de información e enfoque do proxecto: Por un lado, busca de información relacionada co framework Spark e a súa función dentro do contorno Big Data, para que se utilizaba e os seus obxectivos. Tras isto, probas en local para comprender despregadura, funcionamanento e a forma de traballar con el. Por outro lado, selección das ferramentas coas que se ía facer o proxecto e elección da metodoloxía a seguir, tendo claro que a elexida sería unha metolodoxía áxil para conseguir adaptar a forma de traballo ás condicións do proxecto.

2. Análise de ferramentas dos campos tratados: Recopilación de información acerca de ferramentas utilizadas na actualidade para tarefas relacionadas co ámbito do proxecto. Algunhas delas foron recollidas na sección 1.3.

3. Deseño e implementación da ferramenta: Definición das funcionalidades xerais que proporcionará a ferramenta desenvolta e posterior implementación destas en diferentes iteracións coñecidas como sprint dentro da metodoloxía áxil seguida, realizando os seguintes pasos en cada un dos sprints:

- Deseño: Análise acerca das posibles alternativas para a implementación dos casos de uso asignados á iteración e xustificación das eleccións.
- Implementación: Elaboración do código necesario para conseguir as funcionalidades definidas na iteración.
- Probas: Realización de probas das funcionalidades desenvoltas durante a iteración.

4. Elaboración da memoria: Documentación de todo o proceso levado a cabo para a realización do proxecto.

5. Elaboración dun manual de usuario: Elaboración dun manual de usuario no que explicar as funcionalidades que proporciona a ferramenta e a forma de utilizalas.

Tal e como se mencionou no capítulo 4, a metodoloxía seguida para desenvolver o proxecto foi Scrum, inda que adaptada ao contexto do proxecto. Polo tanto, todos os sprints seguiron un mesmo proceso, comezando polo Sprint Planning para definir as tarefas que se levarían a cabo durante o sprint, continuando coa implementación das funcionalidades seleccionadas coas súas conseguintes probas funcionais e rematando coa revisión do sprint (Sprint Review), no que se lle presentaban os resultados do traballo ó director do proxecto que actuaba como parte interesada. Cabe mencionar que durante os sprints non tiveron lugar os Scrums diarios xa que polas circunstancias non se podían realizar reunións diarias, e foron substituídas pola comunicación a demanda mediante chat e reunións puntuais.

Na figura 5.1 pode observarse o diagrama de Gantt coa planificación inicial do proxecto, proporcionándonos unha vista xeral das tarefas que se levarían a cabo xunto coas datas aproximadas e duracións para conseguilo, estimando así unha duración total de 8 meses. Pódese observar que o proxecto, ademais dunha fase para a aprendizaxe do ámbito do procesamento de datos distribuídos e a súa utilidade, se planificou en 6 sprints diferentes que dan como resultado a ferramenta de líña de comandos obxectivo.

Os sprints sucédense de forma lineal, marcando a finalización dun o inicio do seguinte, pero non a todos se lles estimou unha mesma duración, xa que por exemplo a implementación inicial do subcomando ”submit” considerouse que necesitaría máis tempo de dedicación debido ás moitas decisións de implementación que conlevaba o seu desenvolvemento con respecto a outras iteracións. Mencionar ademais, que tal e como vemos no diagrama, a tarefa dedicada á escritura da memoria, ”Elaboración memoria”, levouse a cabo de forma simultánea ás demais tarefas, comezando ao mesmo tempo que o sprint 1, xa que se considerou beneficioso ir documentando o desenvolvemento ao mesmo tempo que este ía avanzando.

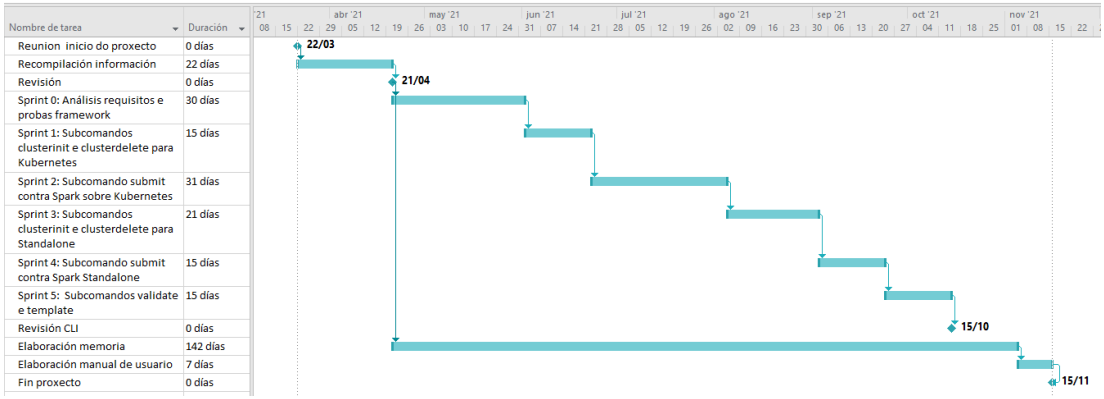

Figura 5.1: Diagrama de Gantt inicial

5.2.1 Custos estimados

Xunto coa planificación do proxecto, tamén se realizou unha estimación dos custos totais asociados á elaboración deste, diferenciando entre:

- Recursos materiais: Este tipo de recursos non supuxeron ningún custo engadido para o proxecto, xa que o software utilizado (sección 3) é de código aberto e en canto ao hardware só é necesario un ordenador (as ferramentas utilizadas soportan distintos SO polo que é indiferente) co cal xa se contaba previamente, polo que non o contamos como custo a engadir.

- Recursos humanos: Foron 2 persoas as partícipes deste proxecto. Por un lado o titor, que desenvolveu o rol de director de proxecto software e o alumno, que desenvolveu un rol de programador Python junior, calculándose os costes asociados a estes recursos facendo unha aproximación baseada en 2 estudos salariais ([18] e [19]) realizados en anos diferentes e enfocados a distintas comunidades autónomas de España.

Baseándonos nos 2 estudos salariais referenciados, tendo en conta que os salarios no sector TI aumentaron dende os anos 2015-2016, sabendo que o estudo do 2021 recolle datos doutras comunidades autónomas que non son Galicia, e que según os convenios se traballan aproximadamente 1750 horas no ano, realizouse unha aproximación para acadar os soldos medios dos 2 roles desenvoltos polos participantes do proxecto, estimando 10€/hora (18.000€ anuais) como soldo medio para o programador Python e 27€/hora (48.000€ anuais) de soldo medio para o director de proxecto software. Ademais, en canto ás horas adicadas á semana por cada integrante, estimáronse 14 horas/semana para o programador, mentras que para o director de proxecto estimáronse 4h en cada un dos sprints tendo en conta reunións e xestión do equipo de desenvolvemento.

Con isto, como os custos dos materiais necesarios contanse como inexistentes debido a utilizar unha linguaxe de programación/tecnoloxías de código aberto e posuír xa os recursos hardware necesarios (ordenador, independenza do SO), os custos totais estimados para o proxecto foron a suma dos custos dos distintos participantes do proxecto, en total 3.728€.

Recurso	Rol	Custo (€/hora)	Tempo a adicar (horas)	Total (€)
Adrián Rodríguez Couto	Programador Python Junior	10	308	3080
Xoán Carlos Pardo Martínez	Director de proxecto	27	24	648
Proxecto				3728

Táboa 5.1: Estimación inicial do custo total do proxecto.

5.3 Sprint 0: Formación sobre o framework Spark

A pesar de ser certamente diferente ao resto de iteracións e non conlevar implementación de código en si e tampouco aportar como resultado un produto entregable, decidiuse considerar este período de tempo como un sprint xa que aportou unha planificación, filtrado de obxectivos e aprendizaxe crucial no transcurso do proxecto.

Este primeiro sprint identifica o inicio do desenvolvemento do proxecto, e comeza cunha reunión na que se expoñen as funcionalidades desexadas para a ferramenta desenvolta e a conseguinte análise para levalo a cabo. Ademais disto, debido a que se partiu dende o descoñecemento total do campo tratado, dedicouse un período de probas específico ó framework Apache Spark, eixe central da ferramenta a crear, para comprender alternativas en canto ás despregaduras dos contornos necesarios e sobre todo comprender o seu funcionamento, xa que ofrece un abano moi amplo de opcións e funcionalidades e o obxectivo da ferramenta ideada era máis conciso, facendo necesario levar a cabo un filtrado.

Realizouse unha análise de requisitos coa cal se definiron as características operacionais que cumpliría o software a desenvolver xunto coas funcionalidades que debería cumprir e contemplouse tamén a metodoloxía áxil pola que optar para o transcurso do proxecto.

Análise

Os obxectivos propostos neste sprint non tiveron que ver coa implementación de casos de uso como comentamos, senon que foron os seguintes:

- Informarse e comprender o framework Apache Spark, facendo probas de execución.

- Decisión acerca de que tipo de aplicación levar a cabo e o seu método de funcionamento.
- Filtrado das funcionalidades concretas de Apache Spark que interesa máis automatizar coa ferramenta.
- Estimación inicial da duración do proxecto e do custo que pode acadar este, definindo tamén as tarefas que o conforman.

Toma de decisións e desenvolvemento

En canto á toma de decisións, durante esta iteración, decídese que a ferramenta deseñada non vai contar con GUI e vai ser unha CLI por varios motivos explicados na sección 5.1 e que, tomando como exemplo outros softwares adicados a tarefas similares, vai funcionar principalmente mediante arquivos de configuración, con formato .INI, nos que se lle especificará a configuración necesaria e desexada para o funcionamento correcto dos distintos subcomandos ofrecidos.

Por outra parte, unha vez xa se realizou o estudo do framework tomáronse varias decisións acerca de aspectos específicos aos que ía dar soporte a ferramenta creada en base ao común que fose o seu uso e o potencial interés que lle denotamos. Entre estas decisións podemos destacar que:

1. Dentro dos distintos modos de procesamento de datos que soporta Apache Spark (en vivo ou por lotes) decidiuse que a ferramenta a desenvolver vai dar soporte única e exclusivamente ó procesamento de traballos por lotes.
2. Apache Spark conta con 2 "modos de despregadura" os cales especifican onde se executa o proceso "driver", modo cliente e modo cluster, e decidiuse que a ferramenta só dea soporte ao modo cluster, o cal implica a execución do proceso "driver" nun dos nós workers do cluster.
3. Seleccionáronse as aplicacións Java/Scala como as únicas ás que se lles dará soporte, a pesar de que o framework dispoña de API para traballar con outras linguaxes como Python.

Por último, como ben se mencionou previamente, este sprint non conta con desenvolvemento de código como tal, senon que foi máis a construción dunha base sólida para os seguintes sprints que conforman o proxecto.

5.4 Sprint 1: Despregadura e eliminación de Spark sobre Kubernetes

Unha vez rematado o sprint anterior adicado fundamentalmente á planificación do proxecto e á aprendizaxe acerca do framework Apache Spark, comezouse co deseño e a implementación da ferramenta como tal.

Durante este sprint comezouse xa coa parte de programación do proxecto, implementando as bases do que sería a estrutura da CLI elaborada en Python e a creación dun modelo inicial a seguir para o ficheiro de configuración, inda que neste momento só incluíndo parámetros necesarios para levantar e deter o cluster de xeito sinxelo, á espera de agregar maior complexidade conforme se foran implementando máis casos de uso.

Posteriormente pasouse á implementación dos subcomandos ”clusterinit” e ”clusterdelete” que permitirían realizar cunha soa liña a despregadura ou a eliminación dun cluster Kubernetes en local que poida ser utilizado para a execución de traballos por lotes Spark, e con estas funcionalidades implementadas e probadas rematou a iteración.

Análise

As principais tarefas a analizar e desenvolver durante este sprint foron:

- Deseño da estrutura básica da CLI.
- Definición do formato que seguirá o ficheiro de configuración utilizado pola ferramenta e selección de parámetros permitidos para os 2 casos de uso implementados nesta iteración.
- Implementación do subcomando ”clusterinit”: Despregadura automatizada dun cluster Spark sobre Kubernetes en local a través da CLI.
- Implementación do subcomando ”clusterdelete”: Facilidade para a eliminación do contorno Kubernetes despregado localmente coa propia ferramenta para peche de ciclo de traballo.

Toma de decisións e desenvolvemento

Unha vez se decidiu traballar con Python, comezou o proceso de buscar como desenvolver unha CLI, xa que proporcionaba unha ampla cantidade de alternativas á hora de facelo debido

a que conta con moitas librerías para dar soporte a diferentes funcionalidades. Neste proxecto, optamos por utilizar ”Click” [20], un paquete sobre Python nado para crear interfaces de liña de comando altamente configurables, e que se enfoca en facer que o proceso de implementación sexa rápido e sinxelo. Tivéronse en conta outras alternativas, como Docopt [21] ou ”Clint”[22], pero finalmente a elección foi Click debido á súa simplicidade para a aprendizaxe e a súa grande utilidade.

Coa base da CLI implementada, pasouse á implementación dos casos de uso que permitirían levantar e eliminar o cluster Kubernetes local de forma automática a través da ferramenta agregándoos como subcomandos, figuras 5.2 e 5.3.

```
(TFG) PS D:\TFG SPARK\SparkBatchRepo\PySparkSubmit> python .\main.py --help
Usage: main.py [OPTIONS] COMMAND [ARGS]...

  Command-line tool for easy execution of batch jobs with Spark.

Options:
  --help  Show this message and exit.

Commands:
  clusterdelete  Stops the running Kubernetes cluster and deletes all...
  clusterinit    Deploys a local Kubernetes cluster to run your Spark...
```

Figura 5.2: Execución da CLI implementada coa opción de axuda para ver as opcións

```
(TFG) PS D:\TFG SPARK\SparkBatchRepo\PySparkSubmit> python .\main.py clusterinit --help
Usage: main.py clusterinit [OPTIONS]

  Deploys a local Kubernetes cluster to run your Spark batch jobs against

Options:
  --configfile PATH  Path to the configuration file  [required]
  --help             Show this message and exit.
```

Figura 5.3: Execución da axuda acerca do subcomando clusterinit.

En canto á implementación dos casos de uso mencionados houbo que tomar decisións sobre como realizar a despregadura/eliminación do cluster mediante Python, e a decisión final foi realizalo mediante a combinación de Minikube (3.3), ferramenta de liña de comandos que nos permite crear un cluster Kubernetes localmente, xunto co módulo ”subprocess” de Python, o cal nos permite lanzar novos subprocesos e conectar coas suas entradas/saídas estándar para coñecer os resultados das execucións de proceso creado. Coa combinación disto, conségguese lanzar procesos que executan a ferramenta Minikube e configurar o cluster Kubernetes ao noso gusto e cumprindo cos parámetros para a execución da aplicación especificados polo usuario, además de poder eliminalo tamén con dita combinación.

Minikube permite facer a despregadura dos clusters Kubernetes mediante distintos softwa-

res de virtualización, pero outra das decisións do desenvolvemento foi que no noso ámbito, as despregaduras de Kubernetes coa nosa ferramenta só se realizarían con Docker como hipervisor, mais non co resto de opcións permitidas por Minikube de forma nativa. Ademais disto, co obxectivo de poder despregar distintos clusters, Minikube separa as despregaduras por "perfiles", utilizando por defecto un chamado "default", e co obxectivo de facilitar todo para o usuario, tomouse como decisión de deseño que a ferramenta sempre faría a despregadura/eliminación en Minikube co mesmo nome de perfil, xa que non se considera que poder traballar ao mesmo tempo con varios clusters Kubernetes despregados coa ferramenta localmente poidera ser algo fundamental e que aumentara moito a utilidade desta. Todo isto comentado pode verse na figura 5.4, na cal executamos o comando clusterinit da ferramenta e se pode observar que o driver hipervisor utilizado é Docker, xunto co nome de perfil usado de xeito estándar, EasySpark.

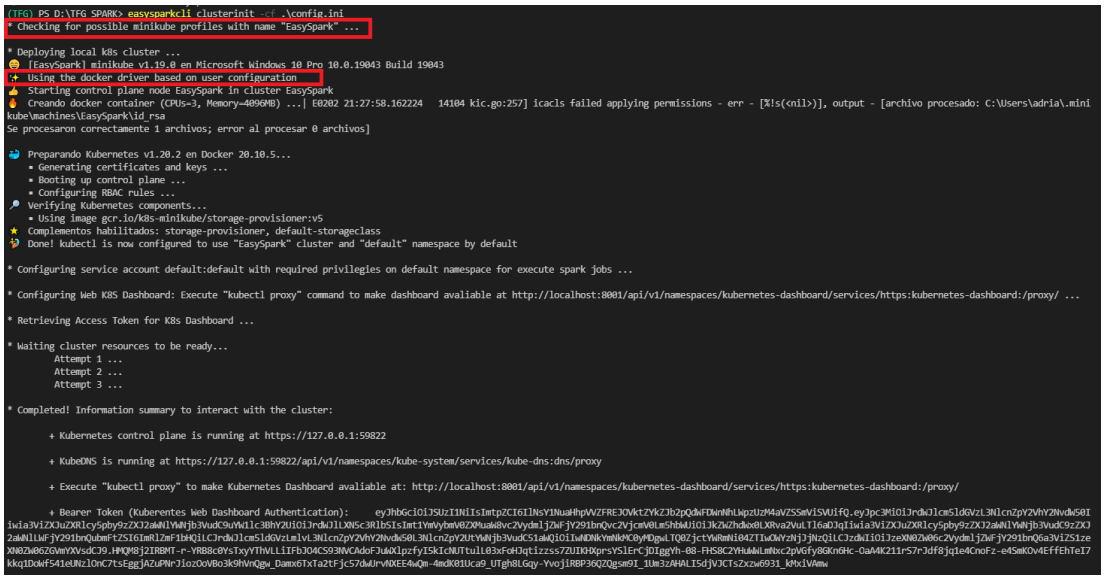

Figura 5.4: Despregadura dun cluster Spark sobre Kubernetes coa ferramenta desenvolta.

Existen outras alternativas que se poderían ter utilizado para manexar un cluster Kubernetes, como por exemplo "Kind" ou "kubeadm", pero finalmente Minikube foi a escollida debido á súa sinxeleza e facilidade de uso, ademais de xa ter traballado con ela en materias da carreira.

Xunto coa despregadura do cluster Kubernetes, decidiuse que o subcomando "clusterinit" tamén configurase de xeito automático todo o necesario para que os usuarios poidesen utilizar de xeito sinxelo unha interface Web (figura 5.5) con información acerca do cluster Kubernetes, a cal terían que activar soamente executando o comando "kubectl proxy" nunha terminal e conseguirían así ter dispoñible en local unha Web dende a cal poderían consultar os distintos

traballos e logs executados no cluster, tendo así outra alternativa á opción de obter os logs mediante a terminal ou nun ficheiro de texto. A información de como habilitar dita interface web, a URL na que se publica e o token a utilizar para poder autenticarse en dita Web son facilitados ó remate da execución do subcomando *clusterinit* tal e como podemos ver na figura 5.4 anterior, na parte do resumo da información.

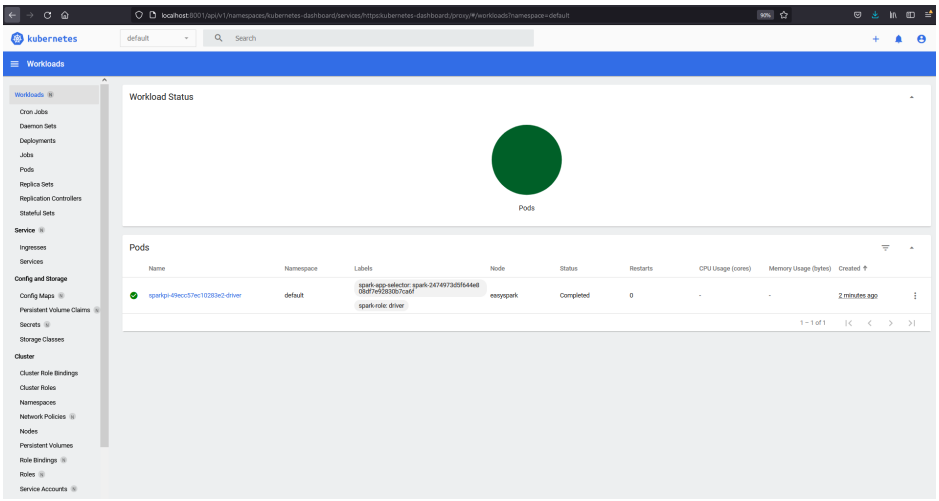

Figura 5.5: Interface Web do cluster Kubernetes configurada polo comando clusterinit.

Por último, durante a implementación dos subcomandos clusterinit e clustedelete, houbo que definir os parámetros que lles poderían ser útiles a un usuario para personalizar o rendemento do cluster a despregar. Con ese fin definiuse a estrutura do ficheiro de configuración a pasar aos subcomandos como argumento, como podemos ver na figura 5.6. Decidiuse que ambos os dous comandos (clusterinit, clusterdelete) utilizarían a información da mesma sección, a sección ”cluster” da figura 5.6.

```
SCHEMA={
    'cluster': {
        "deploy_type": "k8s",
        Optional('node_cpus'): positive_int,
        Optional('node_memory'): nodes_memory_units,
        Optional('nodes'): positive_int,
    },
}
```

Figura 5.6: Esquema de validación para a sección cluster do ficheiro de configuración.

5.5 Sprint 2: Submit contra Spark Kubernetes

Co remate da implementación das funcionalidades que permitían despregar e eliminar automaticamente o cluster Kubernetes en local, decidíuse que o seguinte obxectivo fora permitir executar traballos por lotes no cluster a través da ferramenta, e iso foi ao que se adicou este sprint.

Para elo, implementouse un novo subcomando, chamado "submit". Como parte desta implementación, tivo que facerse unha elección sobre como facilitarlles aos usuarios o envío de traballos a Kubernetes e a posterior recuperación dos resultados, escollendo as opcións de configuración de Spark que máis lles foran interesar para que fose sinxelo e práctico especificalas, pero sempre permitindo utilizar toda a funcionalidade do cliente "spark-submit".

Análise

As tarefas que se analizaron e levaron a cabo durante este sprint foron:

- Incremento do ficheiro de configuración: Crear a sección correspondente ao submit e definir os valores de configuración que se lle permiten especificar aos usuarios nela co obxectivo de facilitar a execución de traballos por lotes en Spark sobre Kubernetes.
- Implementación do subcomando "submit": Facilidade da ferramenta CLI para executar traballos por lotes en Spark sobre Kubernetes.

Toma de decisións e desenvolvemento

Durante este sprint, comezouse decidindo como implementar o subcomando "submit" para que executase o "spark-submit" nativo do framework Spark, chegando á decisión de que, ao igual que para o uso de Minikube, debido á falta dun módulo que o facilitara especificamente dende Python, a execución se realizaría co módulo de Python "subprocess" que permitiría lanzar un novo proceso que fixera a petición ó cluster. Ademais disto, decidiuse como aportarlle a este proceso todos os argumentos especificados polo usuario, adaptando as opcións do ficheiro de configuración da ferramenta ó formato aceptado por "spark-submit". Para isto, optouse por elaborar un ficheiro temporal que só existe durante a execución do subcomando "submit" no cal se escribirían todas as opcións seguindo o formato do "spark-submit", e especificando a maiores valores por defecto que nos permitirían levar a cabo unha execución completa dende a CLI, como ben poden ser puntos de montaxe entre o equipo local e o contorno virtual para a compartición de arquivos e recuperación de resultados.

Posteriormente, pasouse a darlle forma á nova sección que se incluíu no esquema do ficheiro de configuración (figura 5.7) creado previamente, que acabou recibindo o nome de ”submit” e define as opcións relativas a un ”spark-submit” que poden especificar os usuarios para realizar o envío do traballo a executar no cluster. Este proceso foi temporalmente extenso, xa que houbo que escoller que opcións se permitirían e serían máis útiles, e como darlles soporte no cluster levantado de forma local.

```
SCHEMA={
    'cluster': {
        "deploy_type": "k8s",
        Optional('node_cpus'): positive_int,
        Optional('node_memory'): nodes_memory_units,
        Optional('nodes'): positive_int,
    },
    'submit':{
        'master': master_url,
        'class': nonempty_str,
        'app_jar': app_jar,
        Optional('container_image'):nonempty_str,
        Optional('app_args'): nonempty_str,
        Optional('name'): nonempty_str,
        Optional('jarsdir'):existing_dir,
        Optional('jars'):existing_files,
        Optional('libsdir'):existing_dir,
        Optional('driver_memory'):memory_units,
        Optional('driver_cores'):positive_int,
        Optional('executor_memory'):memory_units,
        Optional('executor_cores'):positive_int,
        Optional('executor_instances'): positive_int,
        Optional('historylogs_dir'):existing_dir,
        Optional('advanced'):check_advanced,
        Optional('driverlogs_file'):  nonempty_str,
    }
}
```

Figura 5.7: Incremento do esquema de validación para o ficheiro de configuración.

Por exemplo, en canto á configuración traballando con Spark sobre Kubernetes, é necesario que no momento de facer o submit, os usuarios proporcionen unha imaxe Docker que poida ser despregada en contedores que se crean dentro dun Pod e que actúan como drivers

e executors e levan a cabo a execución do traballo por lotes enviado. É por isto que, para evitarlles ós usuarios o traballo de ter que preparar a imaxe Docker válida para isto, se decidiu que a ferramenta utilizara por defecto unha imaxe válida xa preparada e subida a Dockerhub de maneira pública. Non obstante, tamén se lle da a opción aos usuarios de especificar unha imaxe concreta mediante o ficheiro de configuración, que sobrescribirá a usada por defecto.

Por último, para recuperar os resultados da execución dos traballos por lotes, tomouse a decisión de utilizar a librería "kubernetes-client", que permite xerar un cliente en Python para facer peticións á API Kubernetes do cluster, para recuperar os logs dos traballos Spark executados no cluster.

5.6 Sprint 3: Despregadura e eliminación de Spark Standalone

O comezo deste sprint dase cando se remata a facilitación de todo un ciclo de traballo completo na execución dun traballo por lotes en Spark sobre Kubernetes coa ferramenta desenvolta, proporcionando a posibilidade de despregar o cluster, enviarlle os traballos e logo eliminalo usando os 3 subcomandos implementados.

Con isto realizado, probado e mostrado ó titor (xa que actúa entre outros roles como parte interesada/cliente do proxecto), decídese pasar a unha nova fase, na que se terá como obxectivo proporcionar soporte ao mesmo ciclo de traballo, pero sobre outro cluster manager distinto a Kubernetes dos soportados polo framework Spark, en concreto o coñecido como Standalone. Isto daralles aos usuarios da ferramenta a opción de non ter que depender de ter Docker e Minikube instalados para poder traballar con contedores, e poder realizar o mesmo tendo como única dependencia o framework Spark instalado en local e o software de virtualización VirtualBox.

Sen embargo, durante este sprint o traballo realizado non se adicou a dar soporte ó ciclo completo, senon que ao igual que anteriormente, para esta iteración colléronse como obxectivos as tarefas do Product Backlog que tiñan como obxectivo aumentar a funcionalidade dos subcomandos "clusterinit" e "clusterdelete" para poder traballar con Spark en modo Standalone, e a extensión do subcomando "submit" deixouse para unha iteración posterior.

Análise

Os puntos tratados durante esta iteración foron:

- Extensión do subcomando "clusterinit": Modificación do subcomando clusterinit para

que a maiores da funcionalidade anterior, agora permita despregar un cluster Spark en modo Standalone.

- Extensión do subcomando ”clusterdelete”: Modificación do subcomando clusterdelete para que agora permita tamén eliminar unha despregadura de cluster Spark en modo Standalone.

Toma de decisións e desenvolvemento

A decisión máis importante tomada neste sprint foi a de escoller os módulos Python a utilizar para implementar o subcomando que desprega e elimina automaticamente un cluster Spark en modo Standalone.

Analizáronse varias opcións, como por exemplo a de utilizar o módulo subprocess como se veu facendo en iteracións anteriores para crear as máquinas lanzando procesos que executasen ”VBoxManage”, a ferramenta CLI de VirtualBox, ou lanzar procesos con subprocess que executasen Vagrant, sen embargo, debido a que foi a opción que nos pareceu máis axeitada, sencilla e ”pythonista”, a decisión final foi a de realizar a despregadura mediante o módulo ”python-vagrant”, un wrapper en Python para o executable de liña de comandos de Vagrant, ao cal simplemente lle teriamos que indicar o comando de Vagrant a lanzar e a ruta onde lanzalo.

A decisión foi tomada a favor de Vagrant xa que nos permitía definir todo o necesario para a despregadura local nun único ficheiro (coñecido como Vagrantfile), como por exemplo o SO das máquinas virtuais que conforman o cluster e os recursos hardware destas, a instalación de todas as dependencias necesarias en cada máquina virtual para poder executar traballos Spark ou a comunicación entre as máquinas para formar o cluster Spark en modo Standalone.

Coa decisión de usar Vagrant, e coa axuda de scripts Python e arquivos YAML xerados dende o propio código, reimplementouse o subcomando ”clusterinit” para que no caso de que a despregadura solicitada fora Spark en modo Standalone, se procedese a elaborar un arquivo Vagrantfile que describa o aprovisionamento do cluster Spark, e que dito arquivo Vagrantfile fose usado como entrada para o wrapper ”python-vagrant”, un módulo que nos permitiría executar Vagrant dende Python e iniciar así a despregadura. Unha vez executado o subcomando, o usuario obtén como resposta unha mensaxe con información relativa ao cluster despregado, figura 5.8, na que se lle indica a URL destino a especificar para o envío de traballos xunto coa URL na que se despregou a interface de usuario do cluster Spark, figura 5.9, na cal pode consultar o estado das tarefas e logs da execución, que tamén pode recuperar mediante a ferramenta.

```
* Completed! Information summary to interact with the cluster:

        + Spark Standalone Master waiting for submit jobs at "spark://172.28.128.150:7077"

        + Spark Standalone Cluster WebUI availiable at "http://172.28.128.150:8080"
```

Figura 5.8: Información relativa ó cluster na saída do subcomando ”clusterinit”.

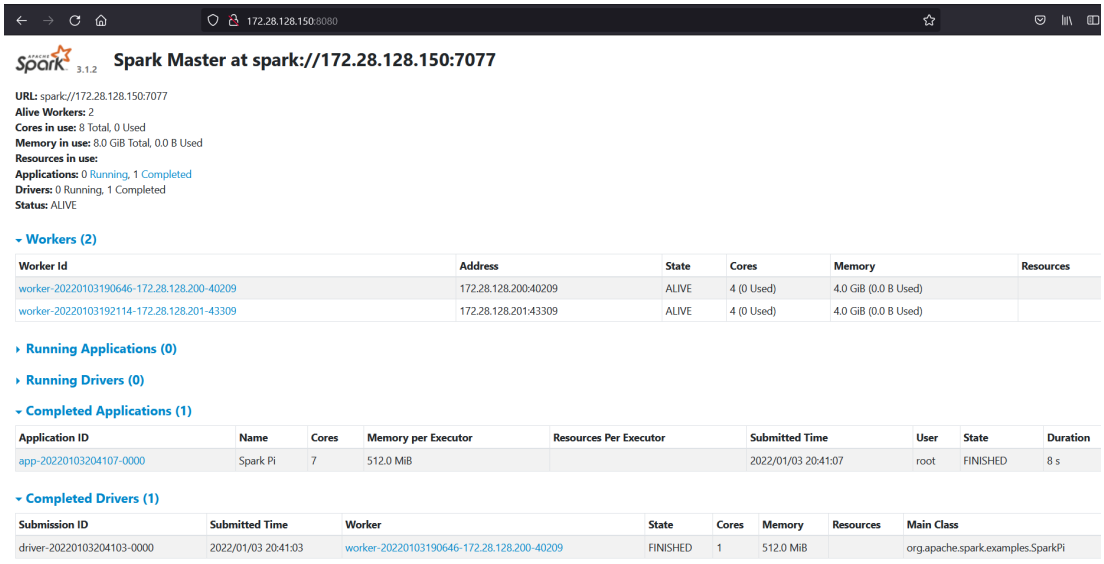

Figura 5.9: Interface web do cluster Spark

Por último, para a extensión da funcionalidade do subcomando ”clusterdelete” para a eliminación do cluster Spark en modo Standalone, fíxose tamén a través do mesmo wrapper xa que Vagrant é unha ferramenta que facilita tanto a despregadura como a eliminación do contorno despregado, o que corrobora que dita elección de implementación foi axeitada para os casos de uso deste sprint. Desta maneira o subcomando ”clusterdelete”, que se apoia na mesma sección do ficheiro de configuración que o clusterinit xa que só necesita coñecer o tipo de cluster despregado, permite eliminar o contorno virtual despregado e todos os arquivos relacionados executando un só comando.

5.7 Sprint 4: Submit contra Spark Standalone

Esta iteración foi unha das máis curtas grazas a que certa parte do traballo podía resolverse de forma sinxela rapidamente ao coincidir cun caso de uso anterior. Unha vez comprobado o funcionamento correcto dos comandos implementados no sprint anterior, deuse paso a este novo sprint, que estaría dedicado a adaptar e extender as funcionalidades do subcomando ”submit” xa existente para que se poida utilizar tamén para enviar traballos a Spark en modo

Standalone e non só sobre Kubernetes como previamente. Como algunhas das opcións de configuración xa coincidían, poideron reutilizarse sen facer moitos cambios sobre o código xa existente.

Aínda así, houbo que facer cambios no esquema de validación aplicado sobre o ficheiro de configuración, xa que as opcións soportadas polo spark-submit nativo son diferentes dependendo de se se usa Kubernetes ou o modo Standalone.

Por outro lado, neste sprint tamén se abordou o problema de como compartir, entre o sistema de ficheiros do computador anfitrión e o cluster virtual, os ficheiros de entrada e saída dos traballos por lotes.

Análise

As tarefas do product backlog escollidas para tratar durante este sprint foron:

- Incremento do ficheiro de configuración: Readaptación da sección ”submit”, debido a que ao engadir un novo tipo de cluster é preciso facer algúns cambios en certas opcións de configuración.
- Extensión do subcomando ”submit”: Reimplementación do subcomando ”submit” para que agora poida traballar tamén con clusters Spark en modo Standalone, ademais do modo Kubernetes soportado en iteracións anteriores.

Toma de decisións e desenvolvemento

Durante este sprint, que foi o máis curto de entre os dedicados á implementación, a decisión principal que se tomou foi a de como permitir a compartición de arquivos entre o contorno local e o contorno virtual despregado de modo que fora transparente para o usuario, da mesma forma que se fixo con Spark sobre Kubernetes. Para elo, e co obxectivo de manter un fluxo de traballo semellante ao caso de Kubernetes, decidiuse que a compartición de arquivos da execución do submit entre o equipo anfitrión e o cluster Standalone virtual se realizaría a través da ruta ”$HOME/.easySparkTool” do equipo anfitrión, aproveitando que dita ruta é accesible tamén dende o cluster virtual. O feito polo que dita ruta é accesible tanto dende o equipo local como do contorno virtual débese a que Vagrant monta por defecto a ruta local sobre a que se realiza a despregadura sobre o contorno virtual, e o subcomando clusterinit da ferramenta prepara todo o necesario para realizar a despregadura nese mesmo path.

Con isto, un dos pasos que realiza o subcomando é copiar os arquivos de dependencias (jars, libs, a aplicación a executar...) da ruta local especificada polo usuario a rutas prepara-

das para ese fin dentro da ruta compartida co cluster, para que así o usuario non teña que preocuparse de como facer chegar os arquivos ós nós que realizan a execución do traballo Spark, e do mesmo xeito, os logs que se obteñen como resultados, son movidos aos cartafois especificados polo usuario se así o especifica no arquivo de configuración.

Ademais, tamén houbo que tocar pequenos detalles en canto á escritura de opcións de configuración no ficheiro temporal que se crea durante a execución do subcomando "submit", xa que aínda que de cara ao usuario se soportan as mesmas opcións para ambas despregaduras, a nivel interno os nomes dalgunhas opcións cambian. Por outro lado, tamén houbo que agregar algunhas novas opcións tanto para o ficheiro temporal do submit como para o esquema do ficheiro de configuración da CLI, como por exemplo a opción "supervise", opción que non existe traballando con Spark en modo Kubernetes, pero que nos pareceu interesante engadir para o modo Standalone, xa que permite que en caso de erro a execución da aplicación se volva a reiniciar de xeito automático.

5.8 Sprint 5: Subcomandos auxiliares e empaquetado da ferramenta

Tras ter conseguido facilitar a execución dun fluxo de traballo completo con Spark (despregadura da infraestrutura, envío de traballo e eliminación da infraestrutura) grazas ós tres subcomandos implementados durante a realización do proxecto, tivo lugar o comezo deste sexto sprint, o derradeiro do proxecto.

O sprint tivo como obxectivo facilitarlles ós usuarios a utilización da ferramenta, para que a poidesen usar con éxito do xeito máis sinxelo posible, e para isto leváronse a cabo varias accións. Por un lado, para facilitarlle ós usuarios a elaboración de ficheiros de configuración que cumpliran co modelo utilizado pola ferramenta, implementouse un subcomando "template" que crearía un ficheiro .ini con todas as opcións que o usuario pode aportar. Deste xeito, o usuario pode utilizar dito ficheiro creado como punto de partida e evita así ter que coñecer previamente as opcións aceptadas e elaborar o ficheiro de configuración dende cero.

Por outro lado, implementouse tamén outro novo subcomando chamado "validate", para que como paso previo a executar os subcomandos de despregadura/eliminación do cluster Spark ou o envío de traballos, subcomandos que requiren obrigatoriamente que se lles proporcione un ficheiro de configuración, se poida realizar unha validación previa de dito ficheiro a aportar como entrada, anticipándose así a un posible fallo dalgún dos outros subcomandos

por non cumprir a configuración aportada co modelo definido.

Ademais, decidiuse estender as opcións do modelo do ficheiro de configuración para que o usuario poidese personalizar o cliente usado para conectar co cluster Kubernetes, solventando así casos nos que os métodos de autenticación da API do cluster poideran estar modificados.

Por último, debido a estar as funcionalidades marcadas como obxectivo xa implementadas, procedeuse ó empaquetamento do proxecto para facilitar a súa distribución e uso, subíndoo a repositorios públicos en GitHub e en PyPI ([23]) para que os usuarios poideran descargar e instalar a ferramenta.

Análise

As tarefas fixadas como obxectivo para este sprint foron:

- Implementación do subcomando validate: Subcomando que permite validar o ficheiro de configuración da ferramenta.
- Implementación do subcomando template: Subcomando que permite xerar un arquivo .ini que serve de modelo para o usuario, contendo a estrutura de todas as opcións de configuración aceptadas pola ferramenta.
- Extensión do esquema do ficheiro de configuración: Adición dunha nova sección que permite personalizar o cliente Python usado no submit a Kubernetes para recuperar os resultados dos traballos enviados.
- Empaquetado da ferramenta e subida a PyPI para facilitar a súa distribución.

Toma de decisións e desenvolvemento

O sprint comezou coa implementación do subcomando ”template”. O obxectivo era que a través de dito subcomando o usuario poidera crear un ficheiro de configuración en formato INI que contase con todas as opcións dispoñibles nun ficheiro de configuración usado como argumento da ferramenta, co fin de poder usalo como modelo para a elaboración do seu ficheiro de configuración propio no cal especificar a despregadura e o envío de traballo personalizado que quere facer. Para isto, Python conta cun módulo coñecido como ConfigpParser que facilita a elaboración de ficheiros de configuración INI, polo que foi escollido para facilitar este proceso e conseguir elaborar o modelo de ficheiro de configuración da ferramenta. Unha mostra adaptada, que non conta con todas as opcións que realmente se engaden no arquivo elaborado polo subcomando template, pode observarse na figura 5.10, mentres que na figura 5.11 podemos ver o as opcións aceptadas por dito comando.

```
1
2       # EasySpark Configuration Template
3       # ===================================
4
5       ;Mandatory section, used both to start/destroy the cluster and to
    run a job
6       [cluster]
7       deploy_type = # k8s OR standalone
8       ; ------- note: optional values for both deploy_types
9       node_cpus = #Default: 2 for Standalone mode, 3 for Kubernetes mode
10      node_memory = #Default: 2048 for Standalone mode, 4096 for
    Kubernetes mode
11      ; ------- note: optional values for standalone deploy_type
12      workers = #Default: 1
13
14      ;Mandatory section for submit subcommand
15      [submit]
16      ; ------- note: required options for submit execution
17      master = #https://172.28.128.150:7077 for Standalone deploy,
    k8s://https://127.0.0.1:XXXX for K8s deploy
18      class = #application main class
19      app_jar = #Path to the bundled jar that includes the application and
    its dependencies.
20      ; ------- note: optional values for both deploy_types, k8s and
    standalone.
21      app_args = #Arguments passed to the main method of the main class,
    if any.
22      name = #Name to show at UI and log data for the application.
23      jarsdir = #Existing local directory path containing possible extra
    jars needed for the execution of the application.
24      jars = #Existing local jars paths needed for the execution of the
    app, separated by commas.
25      libsdir = #Existing local directory path containing possible extra
    libs needed for the execution of the application.
26      libs = #Existing local libs paths needed for the execution of the
    app, separated by commas.
27      ; ------- note: optional values only for k8s deploy_type.
28      container_image = #Container image to use for the Spark application.
    if not specified, a default one is used.
29      ; ------- note: optional values only for standalone deploy_type.
30      supervise = #Boolean value. If its set to True, application
    execution is restarted automatically if it exited with non-zero exit
    code. By default its value is false.
31
32      ;Optional section in case of k8s deploy_type
33      [k8s]
34      api_key = #Provide a specific API Key.
35      api_key_prefix = #Prefix to the specific API Key, default: Bearer.
36      username = #Username for HTTP basic authentication.
37      password = #Password for HTTP basic authentication.
38      retries = #Allowed client connection attempts.
39
```

Figura 5.10: Exemplo de modelo elaborado polo subcomando template.

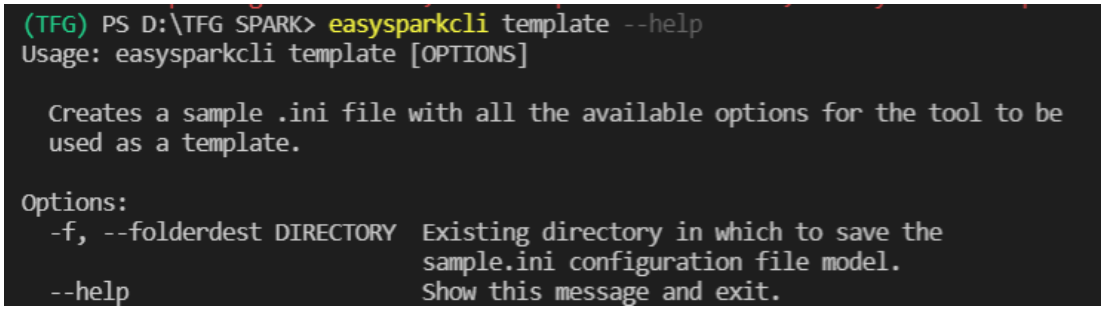

Figura 5.11: Execución da axuda acerca do subcomando template.

Ao rematar a implementación do subcomando template, deuse paso á seguinte tarefa prevista para o sprint actual, que sería implementar un novo subcomando coñecido como ”validate”. Este subcomando tería como obxectivo permitir validar os arquivos de configuración aportados á ferramenta, evitando así atoparse con erros relativos ó formato de dito arquivo na execución doutros subcomandos que reciben como argumento dito ficheiro.

Para a implementación de dita funcionalidade utilizouse unha librería de Python coñecida como Schema, a cal facilita os traballos de validación de estruturas de datos como os obtidos de arquivos de configuración, e que xa se viña usando dende os comezos do proxecto ó tratarse da librería utilizada para definir a estrutura e as opcións dispoñibles nos arquivos de configuración recibidos polos subcomandos clusterinit, clusterdelete ou submit, facendo as validacións nas execucións de ditos subcomandos pero sen dar opción a validar o arquivo antes destas. Como se considerou interesante poder validar seccións específicas dos arquivos de configuración ou a totalidade deste previo á sua utilización como argumento dos outros subcomandos, elaborouse esta utilidade para poder validalos, como podemos ver na figura 5.12. O subcomando permite validar a totalidade do fcheiro de configuración, comprobando todas as seccións que existen nel, ou especificar unha ou varias seccións a validar específicas.

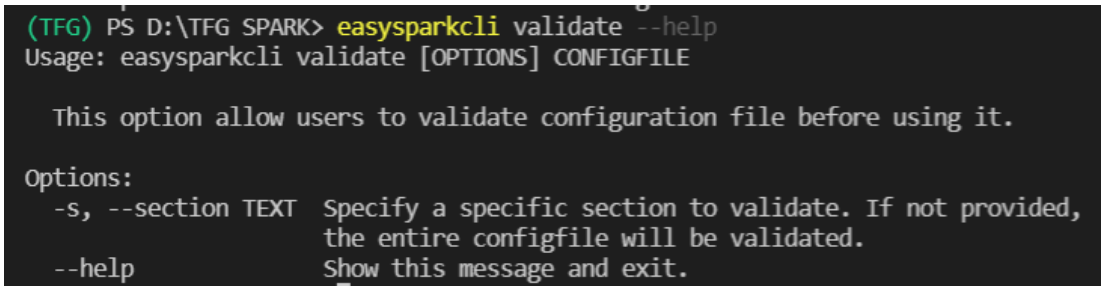

Figura 5.12: Execución da axuda acerca do subcomando validate.

Unha vez implementados os dous novos subcomandos abordados neste sprint, pasouse á adición dunha nova sección nos arquivos de configuración. Dita sección recibiu o nome de "k8s" (figura 5.13) e é unha sección opcional orientada á personalización do cliente de Kubernetes utilizado para conectarse ao cluster Kubernetes Spark, permitíndolles ós usuarios especificar distintas opcións acerca da configuración SSL e a autenticación contra o cluster en caso de que lles sexa necesario e non poidan acceder co método usado por defecto, baseado en adquirir a configuración do ficheiro /.kube/config.

```
SCHEMA_K8S_CLIENT= {
    'k8s': {
        Optional('ssl_ca_cert') : existing_file,
        Optional('api_key') : nonempty_str,
        Optional('api_key_prefix') : nonempty_str,
        Optional('host') : url,
        Optional('username'): nonempty_str,
        Optional('password'): nonempty_str,
        Optional('verify_ssl'): booleanCheck,
        Optional('cert_file'): existing_file,
        Optional('key_file'): existing_file,
        Optional('retries'): positive_int
    },
}
```

Figura 5.13: Esquema da sección k8s engadida cos valores que permite configurar.

Por último, co remate da adición da nova sección comentada, rematáronse as tarefas de implementación de novas funcionalidades para a ferramenta ao conseguir todo o proposto, polo que se pasou ó empaquetado da ferramenta co obxectivo de facilitar a súa distribución e uso para os usuarios. Para isto, Python conta con múltiples alternativas como poden ser PyInstaller [24] ou cx_Freeze [25] que nos permitirían empaquetar o proxecto en arquivos .exe, pero finalmente decidiuse por adoptar a forma estándar de empaquetamento de proxectos Python utilizando setuptools [26] e subir a ferramenta desenvolta como proxecto a PyPI, repositorio oficial para software en Python.

Para isto, preparouse primeiro o proxecto coa estructura de cartafois e arquivos requerida para o empaquetado (figura 5.14), creando os arquivos que configuran o empaquetado como o setup.py, o cal lle sirve á librería setuptools para aportarlle información acerca do noso proxecto e dos ficheiros a incluir, ou o LICENSE.txt, arquivo que lle indica ós usuarios que

instalan o paquete da ferramenta as condicións nas que o poden utilizar.

```
easySparkProject/
├── LICENSE
├── setup.py
├── README.md
├── easysparkcli/
|    └── subcommands/
|    └── easysparkcli.py
|    └── __init__.py
```

Figura 5.14: Estructura de arquivos elaborada para empaquetar coa librería setuptools.

Despois, procedeuse a realizar o empaquetado e realizar a subida do proxecto a PyPI, como podemos ver na figura 5.15, quedándo así dispoñible para que os usuarios o instalen e o utilicen de xeito sinxelo.

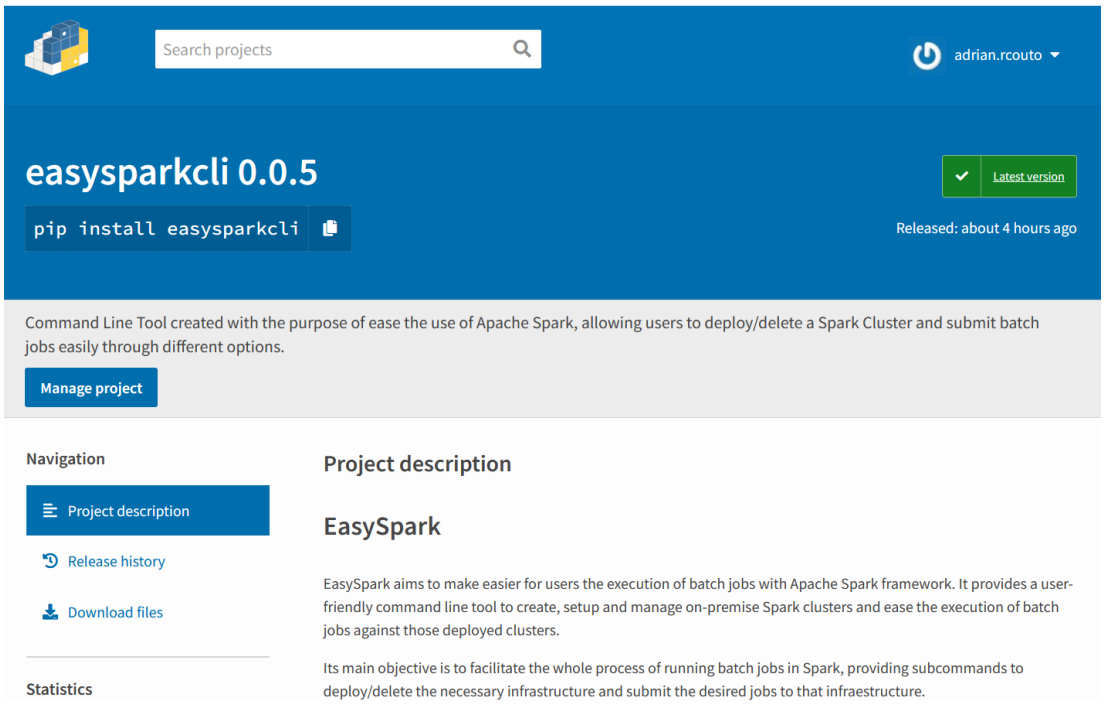

Figura 5.15: Repositorio PyPI do proxecto.

Con isto, unha vez instalado, os usuarios poderían utilizalo introducindo na terminal simplemente o comando ”easysparkcli” e xa verían as opcións de funcionamento da ferramenta, como vemos na figura 5.16.

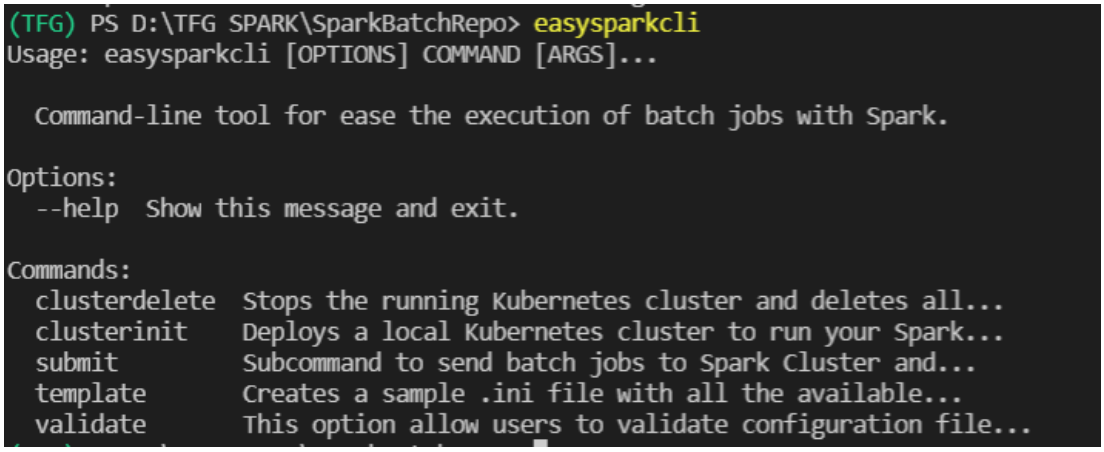

Figura 5.16: Uso da ferramenta dende CLI tras instalación.

5.9 Planificación e custos definitivos

Debido a distintas circunstancias que se deron ao longo do desenvolvemento do proxecto, a planificación non se cumpliu na súa totalidade e tiveron que realizarse readaptacións con respecto ao planificado inicialmente, as cales podemos ver na figura 5.17, diagrama de Gantt que plasma a duración final do proxecto xunto coas súas etapas. En comparación ó planificado inicialmente, a duración do proxecto viuse prolongada. As diferenzas en canto ó planificado orixinariamente foron:

- Aumento de 7 días na duración do sprint 2 en comparación á duración estimada.
- Vacacións do equipo de desenvolvemento dende o 20 de Xullo ata o 16 de Agosto, momento no cal se retomou o proxecto, tal e como se pode apreciar na división da tarefa asociada ó sprint 2.
- Aumento de 6 días na duración do sprint 3 debido a problemas entre a comunicación das máquinas que conformaban o cluster Spark en modo Standalone.
- Pausa do proxecto dende o 9 de Septembro ata o 25 de Outubro por indispoñibilidade do equipo de desenvolvemento.
- Reducción de duración da tarefa asociada á elaboración do manual de usuario.

Debido ós cambios nos tempos adicados ó proxecto con respecto ó planificado inicialmente, tamén se produciron variacións nos custos totais do proxecto, acadando ó final un custo total de 4.062€, contra os 3.728€ estimados orixinariamente.

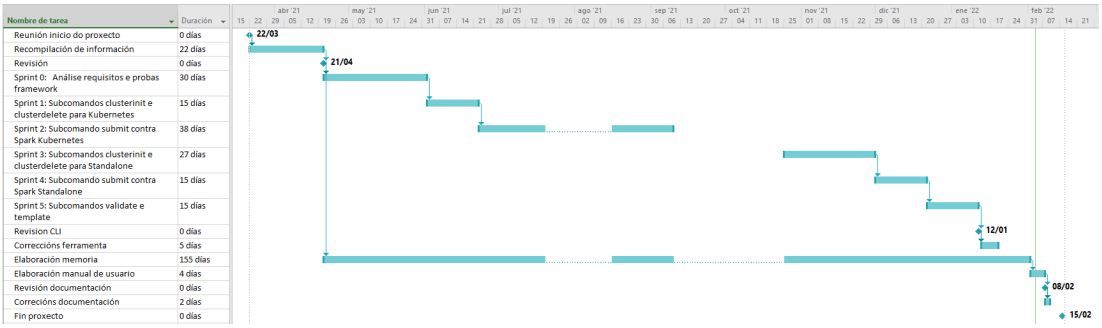

Figura 5.17: Diagrama de Gantt sobre a duración final do proxecto.

Recurso	Rol	Custo (€/hora)	Tempo a adicar (horas)	Total (€)
Adrián Rodríguez Couto	Programador Python Junior	10	336	3360
Xoán Carlos Pardo Martínez	Director de proxecto	27	26	702
Proxecto				4062

Táboa 5.2: Custos finais do proxecto.

5.10 Probas

Co obxectivo de probar que todo o implementado funcionase de forma axeitada, e tal e como marca a metodoloxía Scrum, ao final de cada sprint realizáronse probas do funcionamento dos casos de uso implementados, e nesta sección explícase como se realizaron esas probas.

Python conta cunha ampla variedade de frameworks que facilitan a realización de probas automatizadas, como poden ser PyTest [27] ou UnitTest [28], non obstante, debido ás características do proxecto, ao basearse na despregadura de infraestrutura de xeito automatizado e do envío de traballos mediante spark-submit optouse por facer as probas manualmente.

Todos os comandos da ferramenta fóronse probando de xeito manual tras as súas implementacións nos sprints. Por exemplo, para os comandos clusterinit e clusterdelete, ao tratarse de despregadura e eliminación de infraestrutura realizada con Vagrant ou Minikube, optouse por comprobar manualmente o estado da despregadura co propio software utilizado. Por exemplo, comprobouse que todos os servizos de Apache Spark estivesen funcionando tras executar a creación do cluster e que as interfaces web habilitadas para ambos escenarios estaban funcionando e representando información correcta acerca da infraestrutura despregada.

Por outro lado, con respecto ó subcomando submit realizáronse as probas comprobando

que todas as opcións especificadas polo usuario no ficheiro de configuración se traducían ben ó ficheiro de configuración temporal no formato aceptado por Spark, e que a compartición de arquivos de xeito transparente para os usuarios, que implicaba copias intermedias e montaxe de rutas, fora correcta. En canto aos outros 2 subcomandos restantes, ”template” e ”validate”, optouse tamén polas probas funcionais manuais comprobando que o modelo de ficheiro de configuración contivera todas as opcións soportadas e que a validación dos ficheiros de configuración comprobase todas as validacións soportadas polo esquema.

Capítulo 6

Conclusións e traballo futuro

Derradeiro capítulo da memoria, onde se presentan os resultados e a situación final do traballo, as leccións aprendidas, a relación coas competencias da titulación en xeral e as posibles liñas futuras que se poderían levar a cabo como continuación do proxecto.

6.1 Resultado

Chegado o remate do proxecto, podemos sacar unha serie de conclusións acerca dos resultados deste, facendo unha comparación cos obxectivos marcados nos momentos iniciais.

En concordancia co propósito principal do proxecto, pódese afirmar que o resultado deste foi satisfactorio ao conseguir implementar unha ferramenta de liña de comandos que facilite a execución de traballos por lotes para os usuarios ao proporcionar:

- A instalación, configuración e despregadura de xeito automatizado no equipo local do contorno necesario para traballar tanto con Spark en modo Standalone como sobre Kubernetes, a través da implementación do subcomando "clusterinit".
- A eliminación automatizada dalgún dos contornos que se poden despregar coa funcionalidade anterior, a través do subcomando "clusterdelete"
- O envío e a execución de traballos por lotes ós clusters Spark en modo Standalone ou Kubernetes brindándolle ós usuarios unha forma sinxela de indicar as opcións máis elementais, a través da implementación do subcomando "submit".
- Funcionalidades extra que se engadiron á ferramenta de liña de comandos para facilitar o uso dos tres subcomandos principais da ferramenta e que completan todas as necesidades de utilización do framework Apache Spark, como son os subcomandos "template" e "validate".

Tras o cumplimento de todos estes obxectivos relacionados coa implementación da ferramenta, decidiuse tamén que esta se tratase dun software Open Source, polo que se pode atopar dita ferramenta subida a un repositorio PyPI (https://pypi.org/project/easysparkcli/) cunha licencia *GNU General Public License v3 (GPLv3)* [29], a cal lle garante ós usuarios finais a liberdade de usar, compartir e modificar dito software pero protexendoo de intentos de apropiación que restrinxan as liberdades mencionadas a novos usuarios sempre que a obra sexa compartida ou modificada.

Por último, co obxectivo de brindarlle asistencia ós usuarios da ferramenta, escribiuse un manual de usuario no cal se explican as distintas funcionalidades do software e como utilizalo, o cal pode ser consultado nos anexos desta documentación (A).

6.2 Aprendizaxe e relación coa titulación

A realización deste proxecto considero persoalmente que foi bastante enriquecedora e trouxo consigo unha aprendizaxe continua. Invertiuse tempo en coñecer o framework de computación en cluster Apache Spark, partindo dende o total descoñecemento tanto do seu funcionamento como da súa utilidade, ata conseguir saber executar traballos por lotes sobre ditos clusters e realizar todo o proceso de instalación e configuración dos escenarios necesarios para o seu uso.

A elección de Python como linguaxe de programación a utilizar foi axeitada, xa que se fixo sinxela de aprender e proporcionou moitas facilidades á hora de desenvolver as distintas funcionalidades da ferramenta grazas á gran cantidade de módulos e librerías para diversas funcionalidades coas que conta. É unha linguaxe moi popular e cunha gran comunidade, polo que se conseguía atopar case sempre información útil e de valor para resolver os problemas.

Ademáis, traballouse e aprendiuse acerca da virtualización do hardware e con ferramentas que seguen o paradigma IaC como son Vagrant e Kubernetes, temas que se tratan en materias da carreira, como *Enxeñería de Infraestruturas Informáticas* ou *Administración de Infraestruturas e Sistemas Informáticos*.

Por último, tamén está a aprendizaxe realizada acerca de metodoloxías áxiles, en concreto Scrum, e que garda relación con materias do grao aínda que na mención de Tecnoloxías da Información para a cal aplica este TFG non se lle adican materias completas.

6.3 Liñas futuras

Aínda que os obxectivos propostos foron satisfeitos, unha vez rematado o proxecto xorden novas ideas que se poderían abordar en futuros desenvolvementos, como por exemplo:

- **Linguaxe das aplicacións soportadas:** Actualmente, a ferramenta implementada soporta traballar con aplicacións elaboradas nas linguaxes Java ou Scala, sen embargo o framework Spark soporta executar tamén aplicacións en Python ou en R, polo que se podería engadir soporte para estas 2 últimas linguaxes como unha futura implementación na ferramenta.
- **Modos de despregadura do driver:** A ferramenta desenvolta soporta actualmente só un dos dous modos de despregadura cos que traballa o framework Spark, o modo cluster, polo que como desenvolvemento futuro se lle podería agregar soporte ó modo de despregadura coñecido como cliente.
- **Traballos en vivo:** O framework Spark conta con distintos modos de procesamento de datos, mentres que a ferramenta desenvolta está adicada ó procesamento por lotes. Podería expandirse a utilidade desta ferramenta dando soporte tamén á execución de traballos en vivo con Spark.
- **Contorno Cloud:** As instalacións e configuracións automatizadas de clusters Spark que permite realizar a ferramenta son sobre o contorno local, pero nun futuro poderían automatizarse estas mesmas despregaduras sobre o Cloud.
- **Variedade de hipervisores:** En canto ás despregaduras dos clusters Spark, a ferramenta desenvolta cínguese a utilizar VirtualBox para o caso de desplegar Spark en modo Standalone e Docker no caso de Spark sobre Kubernetes, pero poderíanse aportar como opcións utilizar outros hipervisores para as despregaduras para aportar maior flexibilidade.

Apéndices

Apéndice A

Manual de usuario

EasySpark

Manual de Usuario

Adrian Rodriguez Couto
adrian.rcouto@udc.es

Indice

Requisitos software

Para poder instalar a ferramenta de liña de comandos EasySpark necesítase ter instalado Python (versión 3.6 ou superior) e Pip, o sistema de xestión de paquetes de software escritos na linguaxe Python.

Unha vez instalado, para poder facer un uso exitoso e completo da ferramenta, será necesario contar con distintas ferramentas instaladas:

- *Apache Spark* + *Java*, e establecemento da variable de contorno *%SPARK_HOME* indicando a ruta na que se atopa o framework.
- *Minikube* + *Docker* en caso de desexar despregar clusters Spark en modo Kubernetes, ou *VirtualBox* + *Vagrant* en caso de querer despregar clusters Spark en modo Standalone.

Instalación

De cara ça instalación contamos con 2 alternativas, instalar usando o repositorio de PyPI ou o código fonte subido a GitHub. Para instalar a ferramenta aproveitando o repositorio PyPI teremos que executar:

```
$ pip install easysparkcli
```

Para instalar a ferramenta de liña de comandos EasySpark dende código fonte será necesario clonar o repositorio de GitHub (EasySpark GitHub) ou descargalo en formato ZIP e gardalo no equipo de traballo.

Unha vez feito isto, abre unha terminal na ruta na que se atopa descomprimido o contido de repositorio e executa:

```
$ pip install -e . [--log output.txt]
```

A opción --log non é estrictamente necesaria, pero serviría para almacenar o resultado da instalación nun ficheiro e poder consultalo en caso de ter problemas coa instalación.

Funcionalidades e forma de uso

EasySpark é unha ferramenta CLI que permite de forma sinxela tanto despregar e manexar clusters Spark locais como enviar traballos por lotes para executar sobre ditas infraestruturas despregadas. Para isto, conta con varios subcomandos que nos facilitan ditas tarefas e que para levalas a cabo baseanse na utilización de ficheiros de configuración en formato .INI nos cales se lle aporta á ferramenta os parámetros desexados acerca da acción a realizar.

Para consultar os distintos subcomandos que nos ofrece a ferramenta e a función de cada un destes, contamos coa opción *–help*, a cal se pode utilizar tamén de xeito específico sobre os distintos subcomandos para obter a descripción completa deles ($easysparkcli SUBCOMMAND –help):

```
(TFG) PS D:\TFG SPARK\SparkBatchRepo> easysparkcli --help
Usage: easysparkcli [OPTIONS] COMMAND [ARGS]...

  Command-line tool for ease the execution of batch jobs with Spark.

Options:
  --help  Show this message and exit.

Commands:
  clusterdelete  Deletes deployed on-premise Spark cluster and its...
  clusterinit    Deploys a local Spark cluster to which batch jobs may be...
  submit         Subcommand to send batch jobs to Spark Cluster and...
  template       Creates a sample .ini file with all the available...
  validate       This option allow users to validate configuration file...
```

EasySpark está formada por 5 subcomandos que dan soporte para todo o necesario de cara á execución de traballos por lotes Spark:

1. **Template**: Creación dun ficheiro de configuración .INI que conterá a estrutura aceptada pola ferramenta xunto con todas as opcións dispoñibles, contendo explicacións de para que sirve cada opción. O seu obxectivo é servir de modelo para facilitarlle ós usuarios a creación de ficheiros de configuración válidos.

2. **Validate**: Validar ficheiros de configuración, na súa totalidade ou só certas seccións, como paso previo á súa utilización con outro subcomando para evitar fallos na execución.

3. **Clusterinit**: Despregar e configurar clusters Spark no equipo local de xeito automatizado.

4. **Submit:** Permite enviar traballos por lotes ós clusters Spark.

5. **Clusterdelete:** Eliminación dos posibles clusters Spark despregados coa ferramenta, xunto cos seus ficheiros asociados para non deixar rastro deste.

Ficheiro de configuración .INI

Para despregar e enviar traballos por lotes coa ferramenta, é precisa a creación e configuración de ficheiros de configuración en formato .INI.

Os ficheiros .INI teñen unha estrutura a cumprir. Están divididos en seccións, e dentro de cada sección hai valores que se usan na configuración do programa que xestiona ese ficheiro. Os nomes das seccións indícanse entre corchetes, e inmediatamente seguido van as variables que o programa soporta xunto cos seus valores.

A ferramenta EasySpark traballa con ficheiros de configuración que poden contar con 3 seccións diferenciadas, que serán necesarias segundo o subcomando a utilizar. Ditas seccións reciben os nomes de:

1. **Cluster**

   Sección que recolle os parámetros de configuración do cluster Spark a despregar ou utilizar. É obrigatoria para o uso dos subcomandos *clusterinit*, *clusterdelete* e *submit* e as opcións de configuración que ofrece son:

```
;Mandatory section, used both to start/destroy the cluster and to run a job
[cluster]
deploy_type =
; ------- note: optional values for both deploy_types
node_cpus =
node_memory =
; ------- note: optional values for standalone deploy_type
workers =
; ------- note: optional values for k8s deploy_type
nodes =
```

2. **K8s**

   Sección opcional que só é utilizada en caso de especificarse na execución do subcomando submit enviando traballos a clusters Spark en modo Kubernetes, e que serve para configurar o cliente que recolle os resultados da execución en caso de que a configuración por defecto non permita conectar coa API de dito cluster:

```
;Optional section in case of k8s deploy_type, customize  used K8S client to retrieve results
of the submit. By default, client is configured with ~/kube/.config/ files.
[k8s]
ssl_ca_cert =
api_key =
api_key_prefix =
host =
username =
password =
verify_ssl =
cert_file =
key_file =
retries =
```

3. Submit

Sección que configura o envío de traballos por lotes ós clusters Spark. É obrigatorio que estea presente para a execución do subcomando *submit*, e os valores soportados son:

```
;Mandatory section for submit subcommand
[submit]
; ------- note: required options for submit execution
master =
class =
app_jar =

; ------- note: optional values for both deploy_types, k8s and standalone.
app_args =
name =
jarsdir =
jars =
libsdir =
driver_memory =
driver_cores =
executor_memory =
executor_instances =
logs_dir =
enable_logs =
historylogs_dir =
driverlogs_file =
advanced =
; ------- note: optional value only for k8s deploy_type.
container_image =
; ------- note: optional value only for standalone deploy_type.
supervise =
```

Creación ficheiro de configuración

O subcomando template permite crear un ficheiro .INI con todas as seccións/valores aceptadas pola ferramenta no directorio sobre o que se executa ou sobre unha ruta específica a través da opción -f ou –folderdest:

```
(TFG) PS D:\TFG SPARK> easysparkcli template --help
Usage: easysparkcli template [OPTIONS]

  Creates a sample .ini file with all the available options for the tool to be
  used as a template.

Options:
  -f, --folderdest DIRECTORY  Existing directory in which to save the
                              sample.ini configuration file model.
  --help                      Show this message and exit.
```

Validación ficheiro de configuración

O subcomando validate permite validar o ficheiro de configuración .INI aportado como parámetro, comprobandoo na súa totalidade ou comprobando só seccións concretas que conteña en caso de usar a opción -s ou –section:

```
(TFG) PS D:\TFG SPARK> easysparkcli validate --help
Usage: easysparkcli validate [OPTIONS] CONFIGFILE

  This option allow users to validate configuration file before using it.

Options:
  -s, --section TEXT  Specify a specific section to validate. If not provided,
                      the entire configfile will be validated.
  --help              Show this message and exit.
```

Despregadura local de cluster Spark

A través do subcomando *clusterinit* podemos despregar un cluster Spark en modo Standalone ou Kubernetes, segundo se especifique no ficheiro de configuración a aportar obrigatoriamente coa opción -cf ou –configfile:

```
(TFG) PS D:\TFG SPARK> easysparkcli clusterinit --help
Usage: easysparkcli clusterinit [OPTIONS]

  Deploys a local Spark cluster to which batch jobs may be sent for execution.

Options:
  -cf, --configfile PATH  Path to the configuration file  [required]
  --help                  Show this message and exit.
```

Xunto coa despregadura do cluster, o subcomando *clusterinit* configura unha interface web para a xestión do cluster dende a cal se poderá facer o seguimento dos traballos executados neste. Esta interface é accesible dende a url http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/ en caso de ter despregado o cluster en modo Kubernetes, ou dende http://172.28.128.150:8080 en caso de ter despregado en modo Standalone.

Envío de traballos por lotes ó cluster

O subcomando submit permite enviar aplicacións escritas nas linguaxes Java/Scala ós clusters previamente despregados recibindo toda a información necesaria para o envío a través dun ficheiro de configuración .INI:

```
(TFG) PS D:\TFG SPARK> easysparkcli submit --help
Usage: easysparkcli submit [OPTIONS]

  Subcommand to send batch jobs to Spark Cluster and retrieve execution
  results.

Options:
  -cf, --configfile PATH  Configuration file path  [required]
  --help                  Show this message and exit.
```

A compartición dos arquivos a usar (aplicación e dependencias) é xestionada internamente pola ferramenta, só se ten que aportar a ruta local onde se atopan estes dun xeito correcto, e segundo se especifique no ficheiro de configuración aportado, os resultados da execución serán representados na terminal utilizada para a execución ou almacenados nun ficheiro de texto. Ademais, en caso de solicitalo, a execución de traballos pode xerar tamén ficheiros de histórico, cos cales se poden observar de xeito gráfico as distintas fases polas que pasou a execución da aplicación en caso de contar con un Spark History Server que poida interpretalos.

Eliminación do contorno despregado

Unha vez despregado o cluster Spark e rematada a execución dos traballos por lotes que queriamos realizar, a ferramenta EasySpark aporta o subcomando *clusterdelete* o cal nos permite eliminar o cluster Spark e todos os seus ficheiros asociados a través dun só comando:

```
(TFG) PS D:\TFG SPARK> easysparkcli clusterdelete --help
Usage: easysparkcli clusterdelete [OPTIONS]

  Deletes deployed on-premise Spark cluster and its associated files.

Options:
  -cf, --configfile PATH  Path to the configuration file  [required]
  --help                  Show this message and exit.
```

Ficheiros de configuración de exemplo.

Spark en modo Kubernetes

Ficheiro de configuración coa sección cluster configurada a cal nos permite realizar a despregadura dun cluster Spark en modo Kubernetes de un só nó, con 4GB de memoria e 3 CPUs. Por outro lado, configura tamén a sección submit e deixa a k8s sen especificar (cliente API Kubernetes colle configuración por defecto) para a execución da aplicación "extend_spark_application_example_2.12-0.0.1.jar".

Indícase a clase da aplicación aportada a executar ("class"), a dirección do cluster Kubernetes á que enviar o traballo por lotes ("master"), un nome identificativo para o traballo a enviar ("name"), a cantidade de memoria dispoñible para os executors e drivers ("executor_memory e "driver_memory"), as dependencias da aplicación executada ("jars"), ónde almacenar o ficheiro de histórico e en qué ficheiro gardar os logs do driver ("historylogs_dir" e "driverlogs_file" respectivamente) e por último os argumentos requeridos pola aplicación especificada ("app_args").

```
[cluster]
deploy_type=k8s
node_memory=4096
#node_cpus sen expecificar -> 3 CPUs asignadas
#node_cpus sen especificar -> Despregadura con 1 só nodo

[submit]
class=net.kk17.extend_spark_application_example.CountingApp
master=k8s://127.0.0.1:63985
app_jar=D:\WordCountExtended\target\scala-2.12\extend_spark_application_example_2.12-0.0.1.jar
name=WordCountWithDependencies
executor_memory= 512m
driver_memory= 512m
jars=D:\WordCountExtended\dependencies\play-functional_2.12-2.6.9.jar,D:\WordCountExtended\dependencies\play-json_2.12-2.6.9.jar
app_args=file:C:\Users\adria\Desktop\argsTest\TextoProba.txt file:C:\Users\adria\Desktop\argsTest\output.txt
historylogs_dir=C:/Users/adria/Desktop/spark-k8s/
driverlogs_file=C:/Users/adria/Desktop/spark-k8s/WordCount1-driver-output.txt
#container_image sen especificar -> Utiliza a imaxe docker por defecto almacenada en dockerhub para a ferramenta (adrianrc22/spark:latest).

#[k8s] -> Non especificamos sección k8s, cliente configurarase por defecto cos valores recollidos no fichero ~\.kube\config
```

Spark en modo Standalone

Ficheiro de configuración coa sección cluster configurada de xeito que nos permite realizar a despregadura dun cluster Spark en modo Standalone con 2 nós workers, a maiores do nó master creado de xeito obrigatorio. Cada un destes nós conta con 2048MB de memoria e 2 CPUs.

Para a execución do traballo, aportamos os valores obrigatorios para realizar o submit ("class", "app_jar" e "master") e ademais indicamos como valores opcionais un nome identificativo para a aplicación ("name"), o cartafol no que se atopan as dependencias da aplicación a executar ("jarsdir"), a opción para que o driver se reinicie de forma automática en caso de erro ("supervise"), a configuración de recursos tanto de memoria como de núcleos para os executors e o driver ("executor_cores", "executor_memory", "driver_cores", "driver_memory"), o número de executors a utilizar polo traballo enviado ("executor_instances") e por último os argumentos requeridos pola aplicación a executar ("app_args").

```
[cluster]
deploy_type=standalone
node_memory=2048
node_cpus=2
workers=2

[submit]
class=net.kk17.extend_spark_application_example.CountingApp
master=spark://172.28.128.150:7077
app_jar=D:\WordCountExtended\target\scala-2.12\extend_spark_application_example_2.12-0.0.1.jar
name=WordCountWithDependenciesStandalone
jarsdir=D:\WordCountExtended\dependencies\
app_args=file:C:\Users\adria\Desktop\argsTest\TextoProba.txt file:C:\Users\adria\Desktop\argsTest\output.txt
supervise= True
executor_cores= 2
executor_memory= 1024m
driver_cores= 2
driver_memory= 1024m
executor_instances=1
#historylogs_dir sen especificar -> non se xera ficheiro de histórico
#driverlogs_file sen especificar -> logs do driver imprimidos na terminal usada para a execución
```

Relación de Acrónimos

AWS Amazon Web Services. 3

CLI Command Line Interface. 10, 19

DAG Directed Acyclic Graph. 15

ETL Extract, Transform and Load. 13

GCP Google Cloud Platform. 3

GUI Graphical User Interface. 10, 34

HDFS Hadoop Distributed File System. 14

HPC High Perfomance Computing. 3

IaC Infraestructure as Code. 60

RDD Resilient Distributed Dataset. 14

SQL Structured Query Language. 13

TFG Traballo Fin de Grao. 3

YARN Yet Another Resource Negotiator. 15

Glosario

INI Extensión de arquivo para denotar ficheiros de configuración utilizados por aplicacións dos sistemas operativos Windows e certas aplicacións en ambiente GNU/Linux. 50

Pensamento Lean Medir e obter datos de forma contínua para eliminar ou corrixir aquelas tarefas ou procesos que non aportan valor ao produto ou cliente final, e para potenciar aquilo que si aporta valor, sempre co foco posto na mellora contínua. 26

Bibliografía

[1] K. Schwaber and J. Sutherland, *The Scrum Guide*, 2020.

[2] "Documentación de elasticluster." [En liña]. Dispoñible en: https://elasticluster.readthedocs.io/en/latest/

[3] "Aws parallelcluster - amazon web services." [En liña]. Dispoñible en: https://aws.amazon.com/es/hpc/parallelcluster/

[4] "Apache spark en amazon emr." [En liña]. Dispoñible en: https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark.html

[5] "Azure hdinsight: Hadoop, spark y kafka." [En liña]. Dispoñible en: https://azure.microsoft.com/es-es/services/hdinsight/

[6] "Cyclecloud: Administración de carga de trabajo y clúster de hpc." [En liña]. Dispoñible en: https://azure.microsoft.com/es-es/features/azure-cyclecloud/#overview

[7] "Dataproc | google cloud." [En liña]. Dispoñible en: https://cloud.google.com/dataproc

[8] "Apache spark - unified analytics engine for big data." 2013. [En liña]. Dispoñible en: https://spark.apache.org/

[9] M. C. Matei Zaharia, T. Das, A. Dave, M. M. Justin Ma, M. J. Franklin, S. Shenker, and I. Stoica, *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*. University of California, Berkeley.

[10] DataFlair, "Spark rdd - introduction, features & operation of rdd." [En liña]. Dispoñible en: https://data-flair.training/blogs/spark-rdd-tutorial/

[11] "Documentación de kubernetes." [En liña]. Dispoñible en: https://kubernetes.io/es/docs/home/

[12] Kubernetes, "Minikube documentation." [En liña]. Dispoñible en: https://minikube.sigs.k8s.io/docs/

[13] Hashicorp, "Documentación de vagrant." [En liña]. Dispoñible en: https://www.vagrantup.com/docs

[14] Oracle, "Virtual box documentation." [En liña]. Dispoñible en: https://www.virtualbox.org/wiki/Documentation

[15] P. S. Foundation, "Python 3 official documentation." [En liña]. Dispoñible en: https://docs.python.org/3/

[16] Microsoft, "Visual studio code official documentation." [En liña]. Dispoñible en: https://code.visualstudio.com/docs

[17] E. Abellán, "Scrum: qué es y como funciona esta metodología." [En liña]. Dispoñible en: https://www.wearemarketing.com/es/blog/metodologia-scrum-que-es-y-como-funciona.html

[18] V. Consultores, "Guía salarial sector ti galicia 2015-2016." [En liña]. Dispoñible en: https://es.scribd.com/document/288511179/Guia-Salarial-Sector-TI-Galicia-2015-2016

[19] "Estudio de remuneración 2021." [En liña]. Dispoñible en: https://www.michaelpage.es/sites/michaelpage.es/files/estudio_remuneracion_2021_sp.pdf

[20] "Python click package documentation." [En liña]. Dispoñible en: https://click.palletsprojects.com/en/8.0.x/

[21] D. Organization, "Python docopt package documentation." [En liña]. Dispoñible en: http://docopt.org/

[22] "Python clint (command line interface tools) repository." [En liña]. Dispoñible en: https://github.com/kennethreitz-archive/clint

[23] "Pypi . the python package index." [En liña]. Dispoñible en: https://pypi.org/

[24] "Pyinstaller manual – pyinstaller 4.8 documentation." [En liña]. Dispoñible en: https://pyinstaller.readthedocs.io/en/stable/

[25] "Welcome to cx_freeze's documentation! – cx_freeze 6.10 documentation." [En liña]. Dispoñible en: https://cx-freeze.readthedocs.io/en/latest/

[26] "setuptools 6.50.4 documentation." [En liña]. Dispoñible en: https://setuptools.pypa.io/en/latest/

[27] "Pytest: helps you write better programs." [En liña]. Dispoñible en: https://docs.pytest.org/en/6.2.x/

[28] "Unittest - unit testing framework." [En liña]. Dispoñible en: https://docs.python.org/3/library/unittest.html

[29] F. S. Foundation, "The gnu general public license v3.0 - gnu project." [En liña]. Dispoñible en: https://www.gnu.org/licenses/gpl-3.0.html