Politecnico di Torino

Politecnico di Torino

Master's Degree in Computer Engineering - Computer Network and Cloud Computing

A.a. 2022/2023

October 2023

Master's Degree Thesis

Development and analysis of FinOps processes for a Multi-Cloud environment

Infrastructure as Code support tools for Multi-cloud environments with CI/CD integration

Supervisor:

Prof. SISTO Riccardo

Candidate:

CASTELLANA Luca

Liquid Reply corporate tutors:

Dott. ABBALDO Danilo

Dott. SARAIS Davide

Dott. D'AMORE Matteo

Abstract

Cloud Computing has led companies to rely on remote services instead of build-up their own datacenters. Nowadays many companies are switching to multi-cloud environments to be able to apply a methodological and technical approach to use various cloud-based services in a coherent and integrated modality. However, the bigger is the company, the harder it is to control cloud costs. An innovative approach that can solve this problem is FinOps: a methodology which introduces financial practices and experts into Development and Operational teams.

This thesis work will focus on researching the limitations of FinOps in its current state, developing new proactive and reactive processes that can extend the FinOps practices into the Infrastructure as Code (IaC) approach. After the development and implementation, there will be also an experimental analysis phase regarding these processes, to validate them.

All the developed processes will be built through: Jenkins, an important automation tool, via the creation of scripted automation or CI/CD pipelines; Pulumi, an innovative IaC tool, via the development of Policy as Code scripts; custom scripts written in Python that interact with the Cloud Service Providers through their native APIs.

The processes will extend over multiple cloud providers, so that users will interact with only one system, performing multiple operations. The Cloud Service Providers that will be taken in account are Amazon Web Services (AWS) and Microsoft Azure.

The aim of the primary process developed is the proactive automation and verification of resource tagging, to avoid human errors. The other processes will help the management of a multi-cloud infrastructure via the creation and the control of per project budgets, and the creation of periodic reports after a total check of all the deployed resources with the possibility of performing corrective actions, as destroying some resources which violate the budget or the policy standard.

The experimental validation will verify the correct application and results of the processes developed via statistical comparison between companies that would use these solutions with companies at the actual state. Moreover, a use case study will be presented to explain the correct function of these processes.

The thesis structure is composed by an introduction, the background information, the analysis of the FinOps problem in a multi-cloud environment, the development and explanation of the proactive and reactive processes, the experimental validation, a use case study and the conclusions.

Moreover, the thesis work provides a solid foundation for future studies into the application of FinOps practices in the multi-cloud world, with the possibility of extending it to additional cloud providers, integrating the developed pipelines with existing CI/CD pipelines for IaC or adding other features to this solution.

Table of contents

List of Tables

List of Figures

1 Introduction

The FinOps discipline and the Infrastructure as Code approach, represent innovative methodologies in the cloud computing sector, that together can help in the management of Multi-Cloud infrastructures, leading to the attainment of their maximum benefits.

The FinOps discipline represents the integration of the financial methodology into development and operational approaches. It is an innovative set of practices that is born from the necessity of companies of managing better their cloud costs, especially due to the increasing shift of cloud computing infrastructures from a hybrid cloud approach, where costs were generated by their own infrastructure or a single cloud provider, to a Multi-Cloud approach, where costs depend on multiple cloud providers. The last cited infrastructures represent an innovative alternative for the cloud models, since it is based on the simultaneous application of multiple public Cloud Providers. This approach can bring many benefits for companies, although, its management becomes more difficult as the number of Cloud Providers increases.

For cloud management, an innovative solution is represented by the Infrastructure as Code (IaC) approach. It is a methodology of developing cloud infrastructures via code. It can bring several benefits especially for Multi-Cloud infrastructures since each part will be instantiated via script, leading to a grater control over each component. In the recent years, a new technology has been developed, which is an IaC tool named Pulumi. Its main characteristic is that the code can be written in different generic languages, like Python, Javascript, Typescript and Go instead of a Domain Specific Language.

Since both these technologies are still growing, a study over the feasibility of the FinOps application via Infrastructure as Code approach can be conducted, analysing some aspects of the FinOps practices that can be generalized, to create a base solution that could bring companies to a better management of their cloud spends.

In this thesis work, firstly a presentation of the background about all the technologies and methodologies will be introduced, analysing the cloud computing environments with their datacenters' structure and the innovative cloud category of Multi-Cloud environments, the Infrastructure as Code approach with the presentation of the Pulumi tool, the CI/CD methodology with the presentation of the Jenkins tool and finally the FinOps

methodology; subsequently a theoretical analysis will be conducted over the principal aspects of FinOps practices, focussing over proactivity and automation, and how they can be applied in a Multi-Cloud environment, followed by the development of new processes about the FinOps application for the Infrastructure as Code approach; finally, their experimental evaluation will be performed with also a comparison with some existing tools analysing the limitation and some possible future developments.

The developed processes will focus on various aspects of the FinOps practices based on the FinOps main challenges reported by the FinOps foundation. The main process, will be composed by a CI/CD pipeline, developed via the Jenkins tool, that will perform the automatic application of tags to the infrastructure project, verifying via the development of a Pulumi Policy Pack their correctly application and also verifying other policies that will guarantee a higher financial control before the actual generation of costs. Moreover, the automatic generation of a periodical report via a Python script about a specific typology of cloud anomalies will be also developed with the support of budget management.

Consequently, all these processes will be developed to achieve either proactivity or automation or both, and this will lead to a structural definition of the development process with a reduction in cloud costs, especially for multi cloud environments.

2 Background

Before introducing the developed processes and their analysis, it is important to present some key concepts that will help in the reeding of this work: the Cloud environment and the different types of services available, what are Multi-Cloud infrastructures, the Infrastructure as Code (IaC) approach and how it is used in the Multi-Cloud, the CI/CD methodology, the FinOps approach and how it is used in recent days.

Also, for this thesis work, two different cloud providers will be used: Amazon Web Services (AWS) and Microsoft Azure.

2.1.1 Cloud Computing environment

Cloud Computing is the process of having computing services on-demand, without build-up proprietary servers, so that, based on what each company needs, it is possible to configure a custom infrastructure which will be more scalable, flexible, and not ubiquitous.

There are three main categories of cloud computing [1]:

- *Public Cloud*: all the physical resources are owned and managed by third parties as cloud provider; moreover, companies will pay to use them and will be able to access them only via an internet connection. Thanks to this modality, multiple users will share the same physical resources, optimizing their utilization but still guaranteeing an elevated level of isolation. The major advantages of it are a reduction in the cloud cost, the possibility to focus on the company primary business and high scalability.
- *Private Cloud*: all the physical resources are owned and managed by the company that will use them. In this way the company will have the complete control over each process, but it has to build the datacenter. In most cases it results in higher cloud costs, however it provides more security and control and a more precise customization.
- *Hybrid Cloud*: some resources are used in public cloud, and other in a private one. In this way, a company can obtain the advantages of both the typology, but it will have to manage a more complex infrastructure since data and resources will be divided into different systems. Main advantages of this infrastructure are higher

control, security and management over a reduce part of data and resources and high flexibility and scalability adding resources in the public cloud to the need.

Moreover, in the public cloud category the cloud provider can offer 4 typologies of models of consuming the resources [2]:

- *Hardware as a Service (HaaS)*: the company rent the physical machines. It will have to manage everything except the management of power, cooling, and connectivity supply and the maintenance of the hardware. This solution can bring high configurability and flexibility to the company; however, it increases the cloud cost and the operation needed over the infrastructure.
- *Infrastructure as a Service (IaaS)*: the company rent some virtual component, like storage, virtual machine, etc..., to create a cloud infrastructure for its own use cases. The user can select all the typology of virtualized components, specify the operating system, the typology of data storage, the middleware and the runtimes environment. It can supply high flexibility and scalability of the infrastructure, but it will lead to a reduce control and security over the data and resources.
- *Platform as a Service (PaaS)*: the company rent a Platform where it is possible to deploy some products or application. In this typology fixed hardware, software and infrastructure are supplied to the user, who can only select the platform where he can write, build, test and run codes. The keys benefit of this solution are a faster access to the product and reduced costs for the deploy.
- *Software as a Service (SaaS)*: the company rent a specific Software that will be executed on the Cloud provider's machines. The entire application will be managed by the Provider, leading the company to not have to execute any software maintenance as software updates and bug fixing, or any infrastructure deploy and management. This solution can represent the easiest way to obtain a cloud service since it can be accessed via web without any installation, however it will bring reduction in software personalization, lower control and security over the data and limited performance.

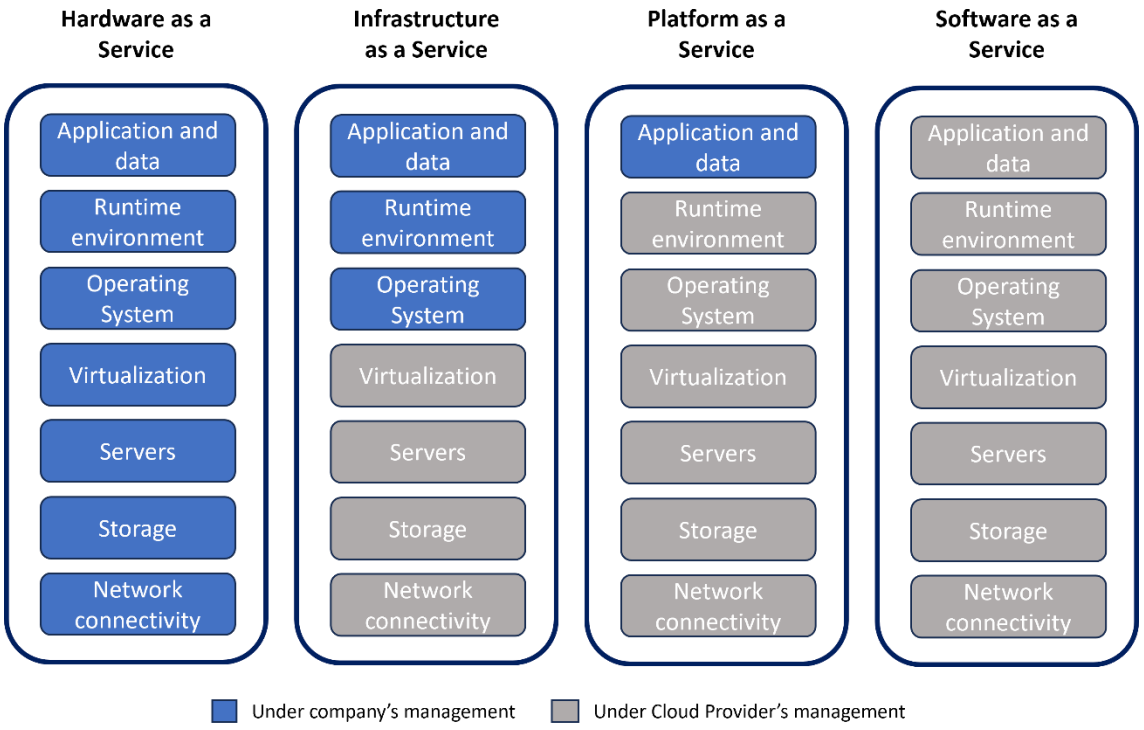


Figure 1 - Cloud Computing models

Since in the private cloud model the company can use servers at will, our focus will be on the IaaS model of public cloud and also on its Multi-Cloud version that will be threated in-depth later.

2.1.2 Cloud datacenters

To provide better services, Cloud Providers structured their datacenters in Regions and Zones. As represented in Figure 2, a Region is a group of at least 3 zones in a close geographic area, that provides a set of replicated services. A Zone is a set of buildings that have in common the same power, network and cooling resources, within which there are the physical servers.

Regions are designed to provide faster, lower latency and localized services. Instead, zones are needed to provide more redundancy and availability to each Region. They are designed to be able to fail, while Regions not: in case a Zone presents any problems, all the services will be transferred to one of the other Zones, so that the provided services will continue working. For this reason, the zones are positioned at a precise geographical distance from each other so that the network latency between them remains acceptable and in the event of a failure or a disaster they are not affected at the same time.

For example, it is possible to consider an e-commerce scenario, where the company deploys its website and databases across a Region. Each service will be deployed in a specific Zone, and in case of this last failure, it will be shifted in another Zone of the same Region to continue to provide the service in the same manner and as quickly as possible.

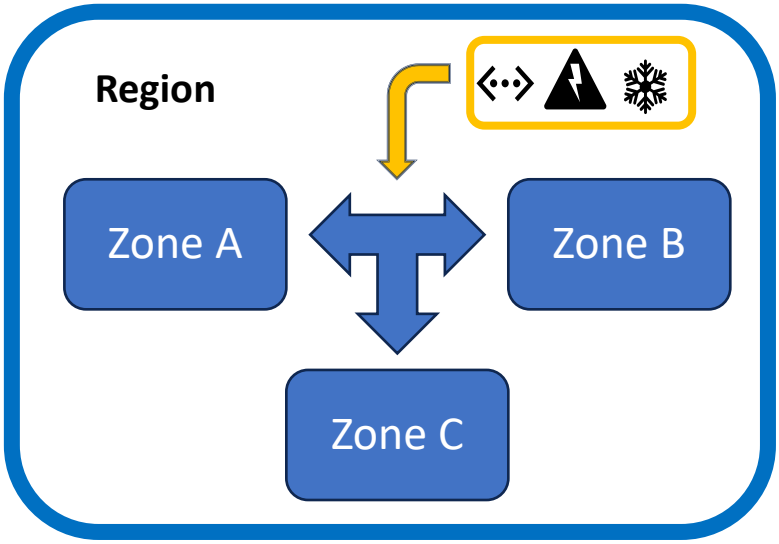


Figure 2 - Region and Zones configuration

Each Cloud Provider offers the possibility to deploy resources in multiple regions and customers will decide where to do it. In this way, each customer can select the best option for his scope, according to various factor like geographical position, network latency, service availability and also commercial agreements. Resources can be either deployed at Zone level, for example virtual machines, or at Region level, as static external IP addresses. However, if region resources can interact with all the other resources, zone resources can directly interact only with the same zone resources. The interaction between regions and the possibility of a region redundancy depends on the Cloud Provider infrastructure. For example in Amazon Web Services regions are designed to be completely isolated, while in Google Cloud region redundancy is allowed [3] [4].

To decide in which region deploy the resources both technical and commercial aspects have to be taken into consideration. For the technical part, network latency is one of the most important components since it will affect the performance of all the infrastructure, while other aspects that have to be considered are: the possibility of a resource to be available in a region instead of another; the dimension and standard usage of the region since if many multiple companies uses the same region it may have reduced performance.

For the economic part two main factors that have to be considered are the resources price and the possibility of saving plans: even if multiple regions belong to the same Cloud Provider, they can offer different resource prices or saving plans. As affirmed in this article of TechTarget [5], each region around the world is subject to different costs about power supply, taxes, duties or operating costs and all of them can modify resource price.

A further consideration has to be done over the expansion possibility of Cloud Providers. Nowadays, many Cloud Provider are sill growing, building new regions that can offer better services to customers. For example, since the European cloud marked is continuing to grow, in the last year, two new regions have been opened in Italy by Microsoft Azure, and Google Cloud [6] [7]. In this way, companies will be increasingly motivated to adopt cloud infrastructures due to highly competitive in a geographic area closer to them which will lead to more convenient costs.

2.1.3 Amazon Web Services and Microsoft Azure comparison

Amazon Web Services (AWS) and Microsoft Azure are both Cloud Service Providers that operate around the world and that in the 2023 according to a research conducted by Statista [8] represent the 55% of the Cloud global market.

Azure operates over 30 different regions, while AWS operates over 32 of them and both offers a various and complete list of services for example data storage, virtual machines or lambda functions. Many of these resources can be properly tagged, to easily identify who deployed them and the project they belong to.

Both offers long term price discounts and sets of tools to manage the infrastructure.

However, a main difference for customers is represented by the way of managing resources: in AWS, a resource can be deployed alone or can be tagged into multiple groups, while in Azure almost all resources have to be deployed into a resource group and can be associated to one resource group at a time. In this way, in the first Cloud Provider, deploying resources is easier and with fewer constraints, while in the second Cloud Provider resources are already grouped and easier to be identified.

2.2 Multi-Cloud infrastructure

As definition, a Multi-Cloud infrastructure is the usage of multiple cloud services from different Cloud Providers at the same time in a cooperative and coherent manner. It can

be seen as an extension of the hybrid cloud where it is possible to include infrastructures based only on multiple public cloud providers, or infrastructures based only on multiple private clouds.

In the last years, as a Gartner research shows [9], companies are switching from an unintentional Multi-Cloud infrastructure, which is the hybrid typology, to an intentional Multi-Cloud infrastructure based on multiple cloud providers in a distributed manner.

This is due to different reasons starting from:

- The possibility of differentiation having a larger set of resources.
- The possibility of choosing the cheapest provider on the market.
- Avoid being tied to a single provider.
- The possibility of redundancy of the resources to have a better resilience.

With Multi-Cloud infrastructure companies can obtain a more personalized infrastructure that can better meet their needs. However, managing this type of infrastructure increases the complexity of the project, and also make more difficult to control cloud cost.

In the same research Gartner defines three phases of the evolutive process of the Multi-Cloud: Multi-Cloud source strategy, Multi-Cloud management and Multi-Cloud architecture. Each one of these phases can have advantages and disadvantages depending on the company's needs, however, after a technical analysis, the benefits to be gained from a Multi-Cloud infrastructure outweigh the risks.

Moreover, a comparison over the five different options of Cloud environment have been carried out. In the research have been taken into consideration Cloud environments with: one strategic Cloud Service Provider; one strategic Cloud Service Provider plus a second preferential Provider for punctual needs; two strategics Cloud Service Providers; two strategic Cloud Service Provider plus a third Provider for punctual needs; no strategic Cloud Service Provider. All of them have been classified in five categories analysing functionality, geographical location, direct costs, management difficulty and data security. The first and last options are considered the riskiest since the first can't supply a good level of functionalities, is limited to a single geographical area and can bring higher direct cost, while the last options will present high difficulties in the infrastructure management and data handling over multiple providers without any preference. The other options have different strengths, but research shows that option with two strategic Cloud Service

Provider plus a third Provider for punctual needs is more balanced, safe, and effective in all the five categories.

	One strategic CSP	One strategic CSP + one for punctual needs	Two strategic CSP	Two strategic CSP + one for punctual needs	No strategic CSP
Functionality	Risky	Good	Warning	Good	Excellent
Geographical location	Risky	Medium	Warning	Good	Excellent
Direct Costs	Risky	Warning	Warning	Good	Excellent
Management	Excellent	Warning	Good	Medium	Risky
Data security	Excellent	Warning	Good	Medium	Risky

Table 1 - Garner research summary about different multi cloud environment

According to a statistical survey performed by Flexera [10] nowadays only the 13% of companies uses complete public Multi-Cloud infrastructure, but according to a Forbes' research, in the next 2-5 years this value will "explode" since the requests of cross-cloud is increasing rapidly.

Even if Multi-Cloud infrastructure can bring several advantages, it present some challenges that should be resolved: the optimization of the infrastructure with resizing options can be extremely harder to be handled; data management and sharing can be more complex with security risks; the integration of actual infrastructure and software environment can represent an obstacle to the modification of the actual company's infrastructure, moreover the Multi Cloud strategy should consider the structural requirements and the company's necessities to create a proper functioning ecosystem.

2.3 Infrastructure as Code paradigm

Generally, companies build and manage Cloud infrastructures by manual processes, with the risk of incurring errors. To solve this problem, a new paradigm has been invented, where the managing and the development of an infrastructure is done through code.

In the Infrastructure as Code (IaC) paradigm, all the infrastructure is described via configuration files, which contains all the definitions, specifications and data necessary to provide the infrastructure [11].

This paradigm can lead to three main advantages:

- An easier way of editing and distribute the configuration thanks to the portability and the shareability of code and the possibility of storing it in online repositories.
- A version control over all the changes applied, with an easier way of roll-back.
- The automation of the setup phase avoiding manual management of the infrastructure.

IaC can be divided into two approaches, declarative and imperative. In the first one the desired states have to be defined and consequently an IaC tool will perform all the changes needed to reach it from the actual state. The latter defines the specific operations that will be executed as a sequence of imperative commands. The declarative approach is an idempotent approach, since the repetition of the instructions will give always the same result, while the imperative approach is not idempotent, since it depends on the previous state of the resources. Both of them can have advantages and disadvantages based on the specific needs for their application: if the desired infrastructure is based on the previous state and it will be more subject to future changes a declarative approach will represent the best choice since a more readable a shorter code is generated, while if the infrastructure have to be built up without considering any previous state the imperative approach will be easier to be applied and faster in the infrastructure coding.

To apply the IaC paradigm, there are different tools companies could use, for example Ansible, Chef, Puppet, Terraform and Pulumi. As reported in this article of Bluelight [12], all of them have different characteristics that can provide advantages and disadvantages. However, the best tools for working with Multi-Cloud infrastructures are the last two

cited since they mainly focus on this typology of infrastructures while the others operate principally with servers' configuration.

For this thesis work, as a IaC tool, will be used Pulumi since it's an innovative and young technology that with respect to its competitor Terraform can offer a grater flexibility thanks to the possibility of using different programming languages instead of a Domain-Specific Language (DSL). Also, there are other advantages, as cited in this article named *"Pulumi vs Terraform"* [13], like the possibility of having a Cloud Native ecosystem Support, the possibility of write both imperative and declarative codes, the possibility of create custom resource via CRUD operation thanks to Dynamic Provider support and higher possibility of testing and validation.

2.3.1 Pulumi

Pulumi is a recent open-source project, born in the 2019, that offers different services from individual free SDK to enterprise program for a fee.

All its tools, as represented in Figure 3, are focused on the Infrastructure as Code approach and the main part of this technology is the Pulumi Engine composed by the Pulumi SDK whose innovation is represented by the possibility of using different existing programming languages instead of a Domain-specific language, and Pulumi CrossGuard a Policy as Code tool where users can create a Policy Pack where they can write function that will validate either individual resources or the full stack. Pulumi also offers other tools that can be used in addition to the Engine, like the Pulumi Cloud with the CI/CD integrator, the Automation API, and a set of Pulumi Packages.

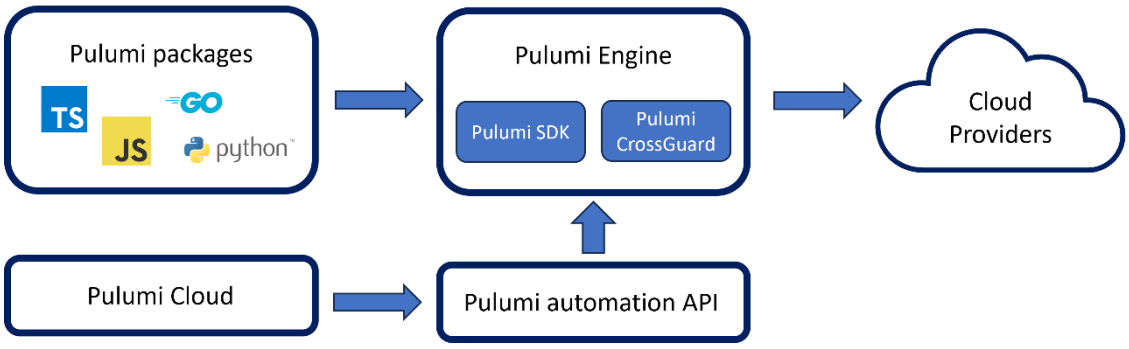


Figure 3 - Pulumi internal structure

The Pulumi Engine is the core of this technology. It will perform the execution and the validation of the Pulumi scripts. To be executed, it needs to be installed on the machine that will perform the operation (i.e. remote server, virtual machine, container) and all its commands can be performed via command line interface [14]. During the deploy, it will interact with the desired Cloud Providers, in order to manage all the described resources. It is composed by a Language Host which is responsible of running a Pulumi Project and a Deployment Engine which will perform the computation to modify the current state of the infrastructure into the desired state defined in the program. The Language Host can be divided in: language executor, a binary program that differs for each language that will be used for Pulumi, and whose name will be *"pulumi-language-<language-name>"* (for example pulumi-language-python) and a language runtime which control the registration of all the resources for the deployment engine. The Deployment Engine will verify the actual state of the deployed infrastructure if present and will perform some decision on how deploy the new resources: it can decide to create a new one, update an existing resource or replace an existing resource with a new one. In the Pulumi Engine the SDK is supplied by the Resource Provider which is composed by the resource plugin and the SDK itself.

Pulumi Cloud is a full-managed platform that can give a complete overview of all the infrastructure created, can store information about resources state and secrets, run remote deployments and also enforce Policy created via the Policy as Code approach. It is automatically used by the Pulumi Engine unless default settings have been changed, however, to be activated each user need to log in. In addition, it can interact with third parties' tool for the authentication allowing an easier organizational account management. The main advantages of this feature are that it can increase Team collaboration, perform some control over Stack permissions for the deployment, supply a CI/CD integration and increase security over all the projects.

Each Pulumi project must have at least one execution stack: a Pulumi stack is an isolated and independently configurable set of deployed resources of a project that are independent from the other stack. In this way, different teams that operate on the same project can use multiple stacks without getting any conflict. Before some Pulumi operation like the deploy, the update, the destroy of a project, Pulumi requires that an active stack has been selected. After the deploy of a project resources, in the program the

pulumi *export* method can be invoked, which will export a specific value or field of a resource in the stack output. When a stack has been selected for a Pulumi Project, a stack configuration file, typically written in YAML, will be automatically generated. An example is represented in the Code 1. This file will keep some configuration that will affect the resource deployment: for example, it is possible to configure the deployment region of the Cloud Provider. Each time the "*pulumi config <desired configuration>*" command will be executed, the desired configuration will be added to this file.

```
1. config:
2.     aws:region: us-east-1
3.     azure-native:location: francecentral
4.     name: test
```

Code 1 - Stack configuration file example

To deploy a Pulumi Project, a series of commands must be executed via a command line interface (CLI): the command "*pulumi new"* will create the new project and after the user can decide to instantiate the new project with a base version based on some example project that Pulumi has made available or some custom project he made; after, the user have to use the command "*pulumi stack *stack Name** " to attach the current project to a development stack; finally the user can use the command "pulumi up" to deploy the infrastructure.

The Pulumi IaC SDK packages are developed across a wide set of Cloud Providers, starting from AWS, Azure, Google Cloud (GCP) and IBM Cloud, to also smaller Provider like Alibaba Cloud, CloudAMQP, ElasticCloud [15]. This wide variety of Providers is important for Multi-Cloud projects, and as this technology is still growing, there are good opportunities of improvement. Its main characteristic is that user can program the infrastructure in multiple languages, for example Python, Javascript, Typescript, Java, Go.

However, Pulumi need time to consolidate and mature. For example, for both the Cloud Providers will be used later in this thesis work AWS and Azure, there are available two versions: a classic one that is at version 6 for AWS and at version 5.49 for Azure, while a native one that is still at version 0.75 for AWS and at version 2.6 for Azure, and in all these versions there are some limitations that will be resolved.

2.4 CI/CD methodology

Continuous Integration / Continuous Delivery or Deployment in a methodology that unify the continuous integration which represent the software development practice via the build, test and merge phases, with the continuous delivery or deployment that represent the phases of release on a repository and deployment to production of the service.

It is a methodology principally applied in the DevOps approach: it tents to unify the Development team with the Operations team with the aim of increase the production and its quality, create a structured and well-defined communication between the teams and increase the transparency and the sharing of information and data. With DevOps, the lifecycle of a product tent to be easier to be managed, since it defines the methodologies to apply to transform an idea into a final product passing from the development environment to the production one.

The CI/CD approach is executed as a cyclic methodology, where, after one part is completed, the other will be again executed. In this way a product is always updated, giving constantly new releases.

The Continuous Integration phase helps multiple developers in working simultaneously on different components of the project, as they will not merge at the same time each new part, but every time one has completed his work and merged with the project those changes will be automatically validated building the project and running the tests. In this way, developers get rapid feedback about what they have done, and also a better versioning of the project is kept in the repository. Continuous Integration relies on some best practices that developers have to use, for example frequent code commit, test categorization, continuous feedback mechanisms, and stage builds. Since it depends on human actions, the frequency at which it can occur can be various, depending on the company and the project that will be developed.

The Continuous Delivery phase focuses on the delivery of the product in a repository after some man-made validations. The delivery will be executed ones the validation of the code is performed, and so when all the test phases are ended. Its scope is to have a codebase always ready for the deployment stage.

The Continuous Deployment phase is the last one that will be executed in this process, and its scope is to automate and make faster the deployment to production of a product. In this way, after a code has passed all the test, it will be automatically released, ready to be used. This steps highly relies on automation processes that will ensure its correct execution. Continuous Deployment will bring some benefits to the production like a faster software delivery, reduced rick after the deploy or after some manual processes and increment teams' collaboration.

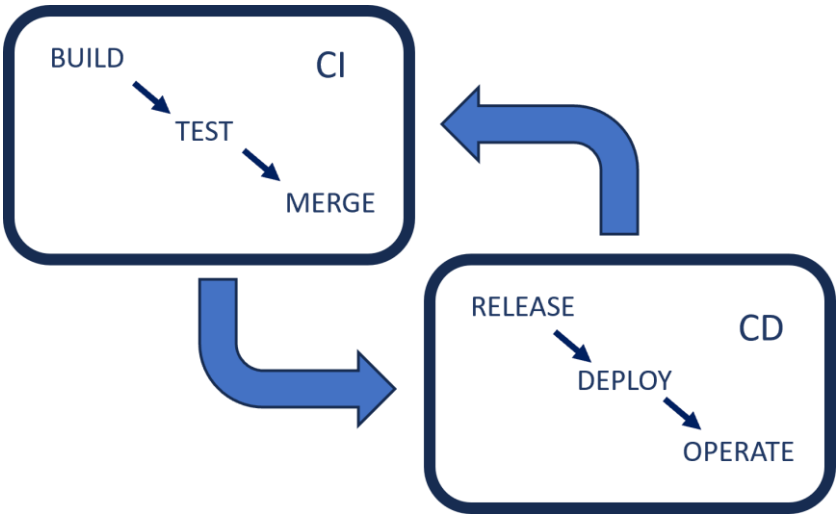


Figure 4 - CI/CD operational schema

Even if both Delivery and Deployment could be associated to the CI/CD methodology, they represent distinct phases that are not necessarily executed together. In fact, the first one typically involves some production-like staging area that increase the time needed to the deploy and that is not automated, while in the deployment this staging area is not required.

To execute CI/CD operations, many tools are available, for example the most popular are Azure Pipelines, GitHub, GitLab and Jenkins. For this thesis work the last one will be used to implement the CI/CD methodology via Jenkins Pipeline, while the test code developed will be saved into a GitLab repository.

2.4.1 Jenkins

Jenkins is an open-source software, written in Java, which offers the possibility of implementing CI/CD workflows. It is executed as a web server that supports the Servlet technology, so can also be used remotely inside a Web Browser.

It was developed to automate some process of the lifecycle of a software, allowing different operation from the execution of multiple stages in an automated way, to the execution via external trigger, for example a code commit on a repository, to implement a more reactive CI/CD approach.

Even if its main scope is to provide a simple CI/CD workflow for software development, this software can be expanded via Jenkins Plugin to extend its functionality, allowing the interaction with third parties' tool like GitHub, GitLab or even some Cloud Service Provider like AWS and Azure, or extending the available operation to be performed inside the pipeline with also Data analysis tool.

Jenkins is structured in a Master-Slave architecture as represented in the Figure 5. The Master is called Jenkins Controller, and it will represent the main interaction point for all the computation. It will store all the information about jobs and pipelines, user credential, plugins interaction and all information about Nodes/Clouds. The slave operators can be divided into static slave that are the Jenkins Node Agent, and the dynamic slave that are the Jenkins Clouds Agent. Nodes Agent are individual local server computation where the master will divide the workload; they can be handled via custom scripts to shutdown and restarts this node to the need. Clouds Agent are server computation built up when some events have been triggered and destroyed after the completion of the execution. They must be executed with a Cloud manager that can supply Virtual Machines or Container, for example Doker, Kubernetes or Amazon ECS. The connectivity between master and slaves can be managed in two different ways: via a Secure Shell (SSH) protocol where all nodes must connect via the port 22 or via the Java Network Launch Protocol (JNLP) where the agent nodes has been initialized with some details of the master and after the master will accept new connection via the JNLP port which is typically the port 50000 but it can be configured.

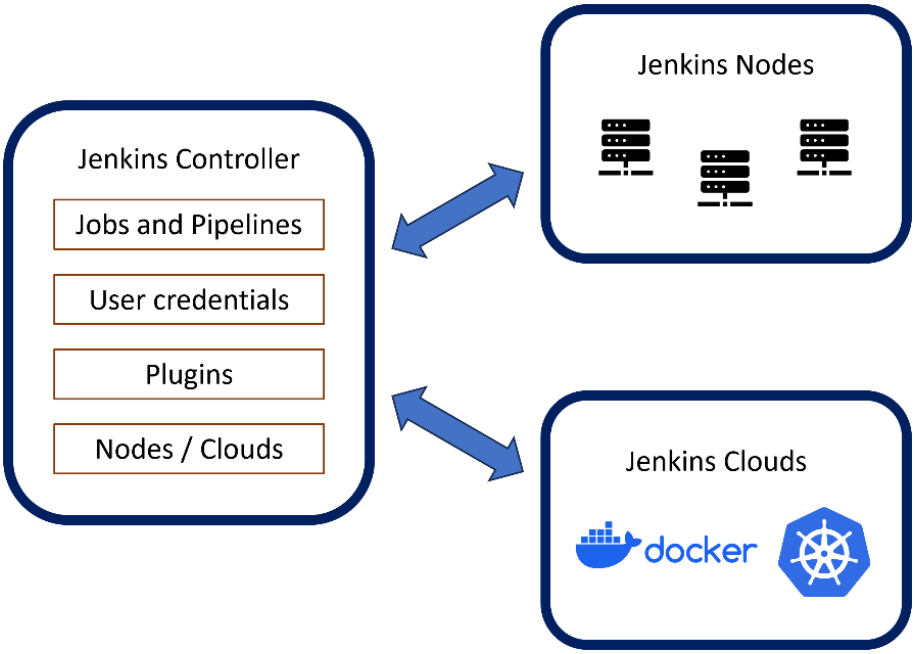


Figure 5 - Jenkins Master/Slave structure representation

In Jenkins, computations are described via Jenkins Pipelines, and they are an automated way of reproducing various steps sequentially.

A Pipeline is composed by stages, and each of them is an individual computation that can fail or pass. In case a stage fails, the entire pipeline will be interrupted. Each stage of a pipeline is composed by steps that are the list of commands that will be executed inside the stage.

To implement CI/CD methodology via Pipelines, each stage will represent a phase of the computation and thanks to the high variety of plug-in available on Jenkins, numerous actions are possible to be performed.

Many of them can be used also to interact with third parties' software, like GitLab or Pulumi, as will be performed later in this thesis work. An example can be represented by the integration of external repository with the pipeline, so that each time there is a modification in the repository the execution of the pipeline is triggered, achieving a complete grade of automation and the execution of the continuous deployment of the product.

2.5 FinOps practices

FinOps is a cloud financial discipline that unify finance with DevOps, which is the combination of development and operations teams, creating new cross-discipline teams. This practice objective is helping companies in managing better their cloud costs, and making financial team cooperate with developers.

Therefore, FinOps is not only a simple way of saving money, but it's a complex discipline that can lead companies to make the best decisions over multiple aspects of the cloud computing.

As described by the FinOps Foundation [16], a no profit organization that offers guidelines and statistic reports over the thematic, FinOps practices are based on six principles, that acts as a guide for teams representing the main goals that need to be reached and adopted:

- Collaboration: financial, business, technology, production teams need to cooperate in real time, improving their efficiency and innovation.
- Cloud business value: all the decisions will not be based only on cloud costs, but on trade-off among cost, quality and development rapidity.
- Resource ownership: each team in the company, will be the direct owner of the deployed resources, being also the direct interested in the accountability of them. In this way there will be decentralized decisions over the costs and teams will be more responsible of how they spend.
- Data visibility: all the FinOps data have to be accessible, processable and sharable at real time, with the possibility of generating periodical reports or spent monitor.
- Centralization: the FinOps team should be centralized, so that it can provide help for all the other team with the FinOps practices and also get advantage from discount optimization due to economies of scale.
- Variable cost model: FinOps team have to use a variable cost model to optimise company's instances and services finding the best quality price.

However, to obtain an elevated level of maturity in FinOps, the simple application of this principles is not enough. An iterative approach is needed to manage at the best variable costs and cloud services: the FinOps lifecycle.

This lifecycle is divided into three phases, that in an iterative way have to be continually managed:

- Information phase: in this first part, the visibility and availability of costs data play a significant role. Since Cloud services are typically on-demand and Cloud Providers offer custom prices and discount, a precise mapping of the cloud spend via tags, account, and budgets forecast can help teams in the deploy of the right resources and help stakeholders in monitoring their investments.
- Optimization phase: in this phase, FinOps team have to optimize the cloud infrastructures, resizing and deleting wasteful uses of resources. Cloud providers wants to make sure that companies keep uses their services, and to do so, offers long term discount plan that can be useful in the optimization phase.
- Operation phase: this is the last phase of the cycle and in this part, companies have to continuously monitor and measure the business objective with the actual state via speed, costs and quality of the infrastructures and services.

The application of this approach will bring companies to achieve a set of capabilities that can be represented in the FinOps domain. The composition of capabilities that the company will achieve will be unique for the organization and can also be a personal target for the increase and improvement of the FinOps practices: capabilities are functional area of activity where financial integration is a key point. Some examples are represented by the Cost allocation, the Accurate Forecasting and the Budget Management.

2.5.1 The Actual State of FinOps

Nowadays many companies are adopting or planning to adopt FinOps practices into their approaches, however, the maturity level that has been reached by them is highly various.

As described into the report redacted by CloudBolt [17], a company that produce software for Hybrid Cloud management, in the 2023 with a survey sample consisting of American, British and Australian companies, the 82% of responders have already a FinOps team, while a 16% of them is planning to create it. However, the 94% or responders thinks that FinOps is a long-term investment, and that in their company more actions are still needed to reach a proficient level of integration. These data are also confirmed in the report generated by the FinOps foundation in the 2022 [18], where the 41% of North American,

the 31% of European and the 7% of Asian Pacific responders affirm that they identify themselves as “Walk” or “Run” in the FinOps maturity level.

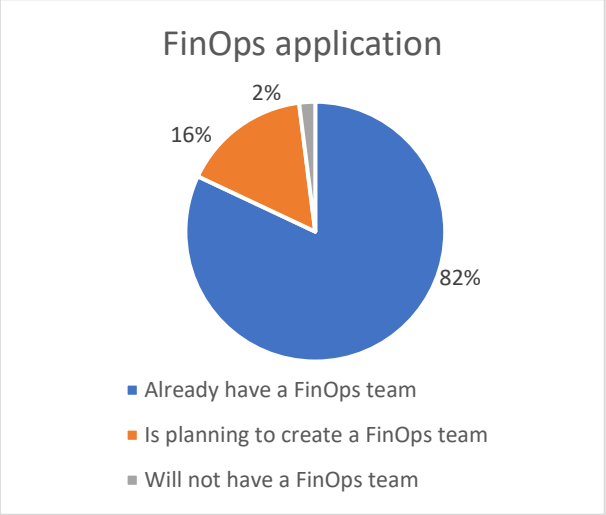


Table 2 - CloudBolt survey FinOps application

The FinOps foundation created a maturity model divided in four categories: Pre-Crowl are teams that are planning to use FinOps but still don’t use it; Crowl are teams that are new to FinOps and that perform little reporting and typically uses reactive approaches; Walk are teams that are able to focus on harder KPIs and that start using proactive and automation tools; Run are teams that are using at the best FinOps practices with complete proactive, integrated and automation tools.

Therefore, it is possible to state that even though companies have invested in creating teams for FinOps, there is still not an adequate level of expertise to get the most out of these practices.

However, a more correct analysis, will consist in the verification of the maturity level of each team through the achievement of the goals in the FinOps capabilities. Consequently, from the last report cited, an analysis over the tools and the methodology applied by these companies can help to understand the limit of FinOps at its actual state and how it can be improved.

In the graph of the FinOps Foundation’s report in Figure 6, there are represented for each category from a list of FinOps capabilities the number of teams divided per maturity level. They majority of companies still have “Crowl” teams that uses reactive tools and approaches with higher difficulties over automation processes. While the higher value of

Run teams is reached in the Cost Allocation category with the 28,1%. Also, Walk team have high statistics over Shared Costs, Managing Anomalies and Cloud Policy, and this can represent a higher interest in these categories from the companies. This interest can be represented by the impact of these capabilities in the actual composition of a company: the modification of the production structure and the team division can be harder to be modified and adapted according to all the FinOps capabilities, while the application of a reduced set of them by a new team inside the company can be more easily applied.

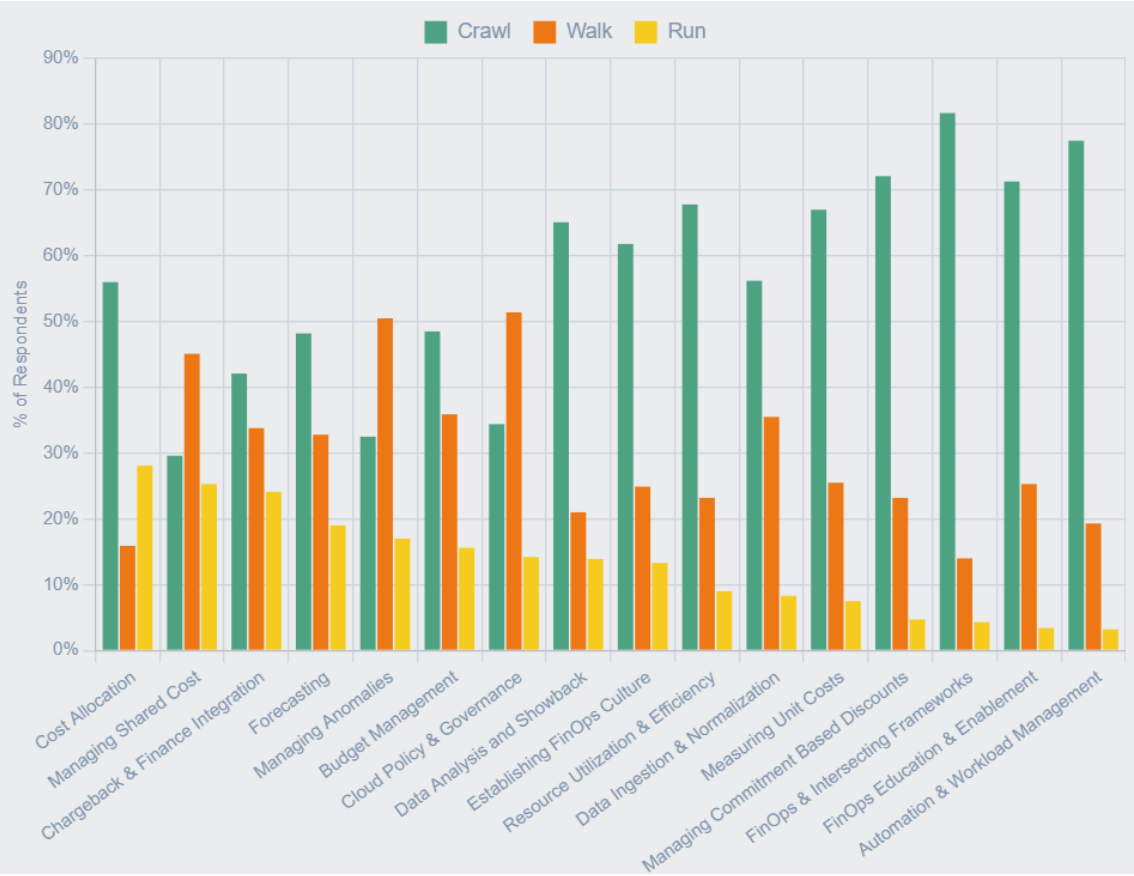


Figure 6 - FinOps Foundation Data: Maturity level of responders over different FinOps practices

Analysing the typology of tools that FinOps teams uses and the grade of automation of them, the lower maturity level of the FinOps team can be also evinced from another graph of the report, reported in Figure 7, that describes the grade of optimization/automation that companies apply across different cloud services. Performing an average of the responses according to the maturity level, the 63,54% of the responders can be considered as Pre-Crowl or Crowl due to the typology of actions that they perform.

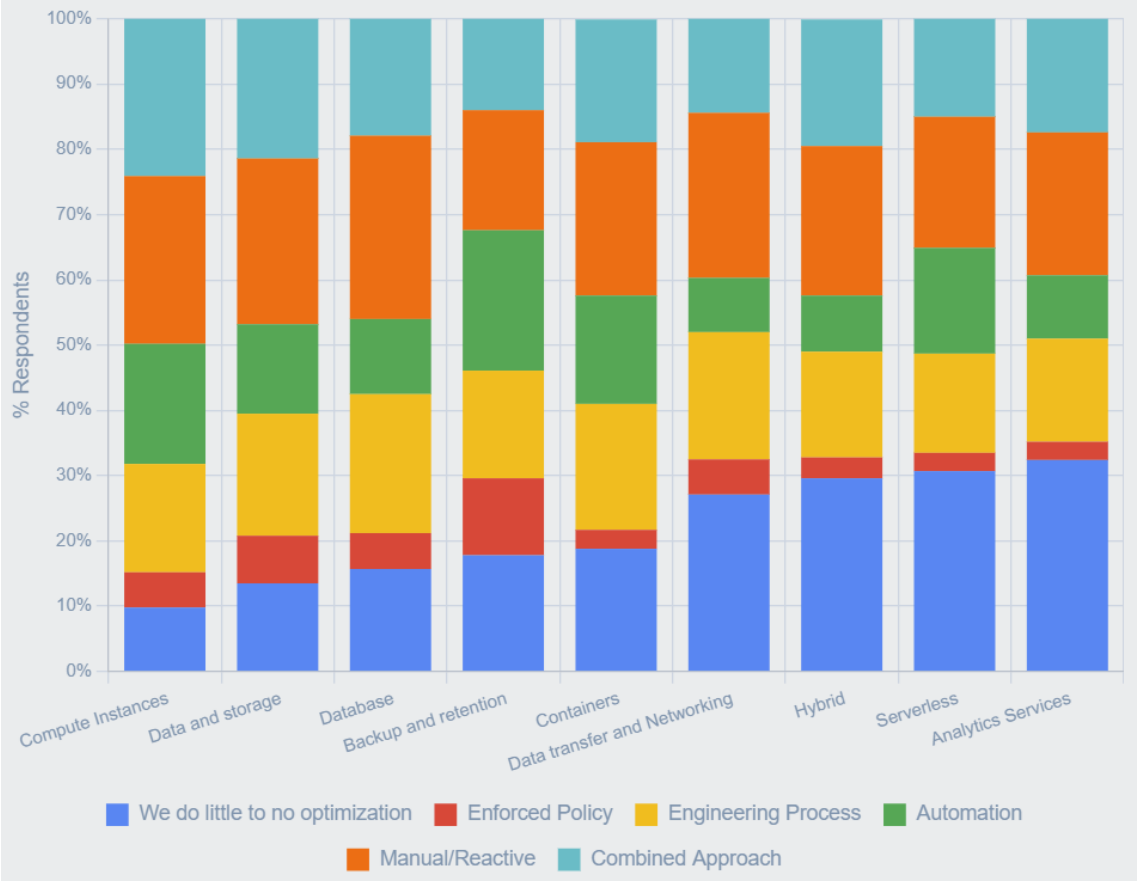


Figure 7 - FinOps Foundation Data: Optimization level over different services

Indeed, automation tools reaches a maximum value of 21,5% in the Backup and retention category and a minimum of 8,3% in Data transfer and networking category, while proactive tools represented by the Enforced Policy and half of the Engineering Process as described in report [19] reaches a maximum value of 20,05% (11,8% of Enforced Policy and 8,25% of Engineering Process) in the Backup and retention category and a minimum value of 10,4% (2,8% of Enforced Policy and 7,6% of Engineering Process) in the Serverless category.

A further important statistic, which can also be evinced from the previous report, is about the ranking of the most important challenges over FinOps according to responders. This statistic can help in understanding which tools and approaches still need to be invested in to improve FinOps practices. Moreover, along with the previous statistics, it can also provide a more precise view over the actual state of the FinOps, allowing to understand how companies are handling these practices and how it will evolve in the future years.

In the Figure 8 are reported the 10 most important challenges according to the report.

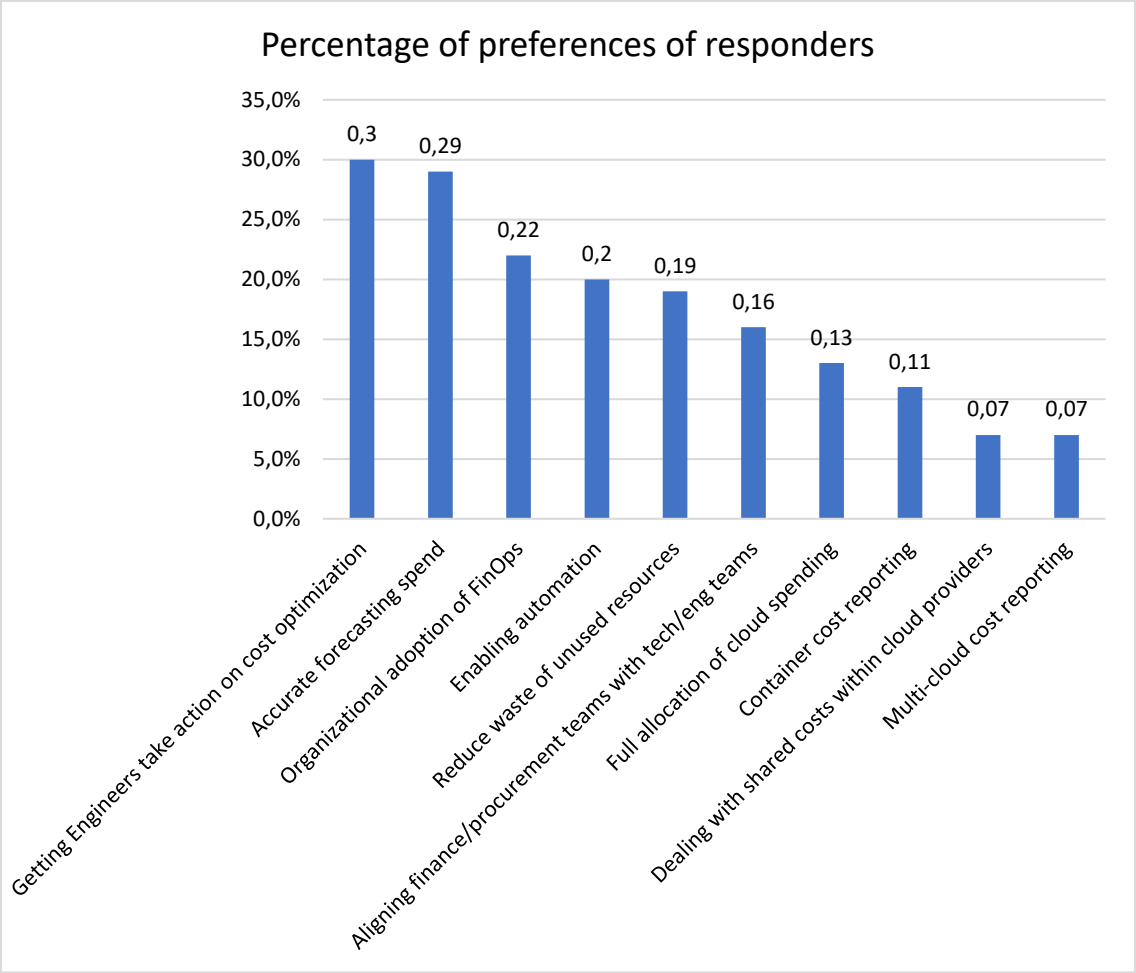


Figure 8 - FinOps Foundation Data: Percentage of preferences over FinOps challenges

This thesis work will focus on proactive and automated processes that will cover many of these challenges: accurate forecasting spend and dealing with shared cost within cloud providers will be covered via Multi-Cloud management; organizational adoption of FinOps and enabling automation will be covered by Jenkins automation pipelines and proactive application of tags to cloud resources; Reduce waste of unused resources and Multi-Cloud cost reporting will be covered via an automatic reporting process over unused and untagged resources.

3 Analysis over innovative FinOps practices and actual existing tools

In this chapter an analysis over innovative FinOps practices will be presented, whose purpose will be the research of the most effective processes that could cover the major FinOps challenges. In addition, a study of some existing tools for FinOps practices will be carried out to understand the limitations of these solutions and how they can be improved.

Following a first generic study over the relation between FinOps and Multi-Cloud environment, a deeper examination will be conducted on the comparison of reactive, proactive and automated processes and also over the Policy as Code approach.

3.1 FinOps over Multi-Cloud environments

An initial analysis can be carried out over how and why FinOps practices are needed in a Multi-Cloud environment.

Nowadays, as cited in the Medium article [20], the majority of companies that still uses single Cloud Provider with a full or hybrid infrastructure are continue to embrace Multi-Cloud environments since this last represent an innovative transformation that could bring many benefit over different aspects.

Companies that use multiple Cloud Service Providers at the same time, typically incur in cloud management problem, since there will be more services that should be managed. Without a proper unified and clear administration, the identification of unnecessary resources that remain active and cost reporting over the complete infrastructure becomes harder to be managed. Consequently, cloud cost can easily increase.

For these reason, FinOps practices are needed in Multi-Cloud environments, as they can offer: a unified vision over all the infrastructure; a complete report of all the cost or of all the wasteful resources; a team and business methodology that can be expanded and perfected without further modification.

To apply FinOps practices in these infrastructures, there are different approaches that depend on the desired result. Their application can vary in both difficulties and adaptation time since they can be used as support for actual standardized processes or as new stand-

alone processes. The more these practices are applied, the more complex but effective they become: to deploy some processes over multiple Cloud Providers, code must be duplicated, or supported by a third parties' tool, for example an Infrastructure as Code (IaC) tool; also, all the financial management, the report generation, the resource location with the relative price and the anomalies detection become more difficult to be handled if not supported by a proper tool. However, once teams adopt these practices using an appropriate structure, costs can come down automatically, getting the most benefit from operating over multiple Cloud Providers.

3.2 Proactive and reactive processes comparison

Reactive and proactive approaches are both solutions that can be used into FinOps practices to manage and reduce costs, report and detect anomalies, optimise the team structure to obtain more benefit.

Both approaches can achieve the desired results, but their methodologies have many differences. Sometimes, for some typology of practices it is possible to apply only one of them, while in other situations it is possible to choose one or the other based on various considerations.

In reactive approaches companies react to events that arise. There could be events that can be handled only with one of them, like the anomalies' detection, and other that can be handled in both ways, for example the check of cloud billing at the end of the month and the execution of corrective actions based on these data.

In proactive approaches companies take action to prevent events. They will execute some operation that will reduce or eliminate the possibility of an event occurring. An example could be budget forecasting, where companies try to understand in advance what will be the billing, so that if corrective actions are needed, they could be executed in time.

When both approaches can be executed, the decision will be based on the complexity of the project and of the company. Proactive approaches tend to be more complex and more expensive to be realized, while reactive approaches are easier to be applied, but less efficient.

Both approaches can have advantages and disadvantages: the reactive approach is more efficient in handling unexpected events and developing innovative solutions, however it

will always be executed after something occurs, trying to mitigate the risks; the proactive approach is focussed on prevention so it can be used for long term scenarios anticipating problems before they arise, however it is not able to handle unpredictable events, requiring long term planning to be more and more effective.

As described by the FinOps foundation [21], with reference to the maturity level, reactive processes are typically used by "Crowl" teams since they are a way of resolve problems when they occur, while "Run" teams have enough experience to be able to set up a proactive methodology in their infrastructure. However, to set up a proactive methodology in the company infrastructure, more time with respect to reactive methodology is needed, and for this reason further investigation on these processes can be carried out to improve the integration with current systems.

For the FinOps approach both reactive and proactive methodologies can be useful to achieve cost reduction. However, to obtain better results proactive processes represents a more complete solution thanks to long term planning and prevention, which can be supported by reactive processes for the mitigation of unpredicted events. Consequently, a focus on proactive processes will be carried out in this thesis work, without excluding those processes that can be exclusively reactive, but that involve added value to FinOps practices.

To achieve proactivity, the main process developed will perform different operation and verification via a Policy Pack, which will prevent from the execution of actions that can cause increasing costs. These verifications will include actual and forecasted budget control and will be able to advertise the user or even interrupt the deploy operation in case a violation of the policy has been detected. In this way, if the budget or its estimates has been exceeded, the FinOps team will take action before the resource deploy will be completed, without increasing the actual costs.

3.3 Automation processes

Automation is one of the hardest processes companies tries to reach out. It can be implemented either on reactive or proactive processes and can bring many benefits to FinOps practices. However, automation processes development could be harder than classical processes, and for this reason it might be more appropriate for larger companies: when a company have to handle a big infrastructure or multiple projects, managing cloud

cost can become more difficult and thanks to automation, FinOps teams can perform more precise and correct operations.

As reported in this article of CloudKeeper [22] automation processes can be divided into five categories:

- *Continuous monitoring*: this automation process aims to give to the FinOps team continuous reports over the costs. It can be created via the detection of unused resources, the detection of resources that can cost less on another Cloud Service Provider or via the detection of out-scaled infrastructures.
- *Tagging and cost allocation*: this automation process aims to reduce tagging error, simplify the detection of unused resources and track effective cloud spending. Tagging is the process of adding some metadata to resources to identify them. For example, some tag can be represented by the project name the resource belong to, which team have deployed it or which person. To define the right tag to be applied, each company should in advance define a tag strategy that will be followed inside all the organization. Typically, this process is one of the most human error prone, since a developer could forget to tag a resource or write the tag in the wrong way. With automation, teams will have a coherent tagging structure, and the FinOps team will easily perform control over all the resources.
- *Alerts and Notifications*: this automation process aims to gives some feedback to FinOps team, so that they can react on time to problems. For example, some budget notification can be set when the limit has been exceeded, or when some infrastructure components stop working correctly. Cloud Service Provider offers many of these systems, however they do not work in a unified way when the infrastructure is deployed in a Multi-Cloud environment.
- *Workflow automation*: this automation process aims to create a proper set of actions and execution workflow within the company so that all the operations can be performed correctly and also can be controlled. Its structure may consist of using several tools together, so that users need only to perform few operations to achieve the final result. Some examples of these processes could be the automated validation and deploy of the infrastructure, or the automated rightsizing of some resources.

- *Multi-teams' collaboration*: this last automation process aims to increase the collaboration between different team, creating automatic report sharing, automatic align over corporate goals and clear and well-defined communication processes.

As described by the FinOps foundation research [23] and reported in Figure 9, automation processes needs to be increased in companies methodologies, since only the 3,2% of organizations can be identified as "Run" in the maturity model with proper and complete automation processes.

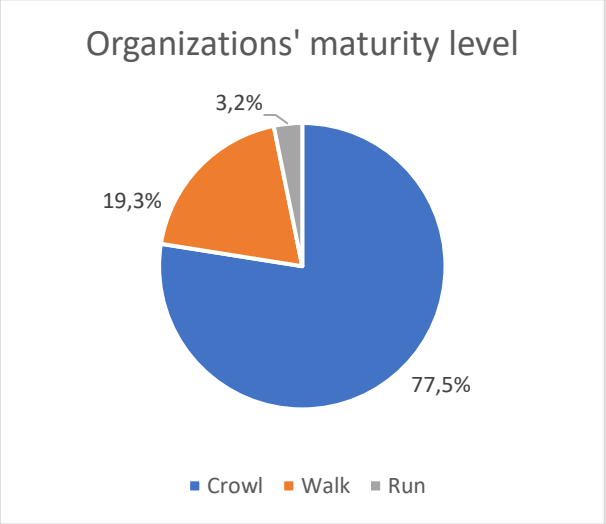


Figure 9 - FinOps foundation report about organizations' maturity level

In this thesis work automation will be obtained in multiple modality: via the definition of a workflow for the deploy of an infrastructure, with also an automatic application of user tags to the resources; via the definition of workflows for users and budget management with the automation of multi-cloud alert definition; via the automatic generation of a periodic report about the untagged resources with relative costs and resources deployed in regions that can cause excessive costs.

3.4 Policy as Code

Policy as Code is an approach to define, share, update and manage policies via the redaction of code. It is a complementary approach to IaC, and while this second focuses on the implementation of the infrastructure, the first one defines a proactive validation to guarantee security, rules and conditions over the resources to be deployed.

Policy as Code is based on the concept of Policy, which can be defined as a rule that code have to respect to continue in the deployment. Any violation can cause either the complete termination of the execution or the generation of an alert.

This approach can bring many benefits to the quality of the infrastructure since it can:

- *Increase team efficiency and speed*: with this approach the Policy application is automated, and so teams will not do it anymore manually. Also updating and sharing policy will become easier and faster.
- *Give more visibility*: defining Policy via code it will be easier to understand how the system will operate.
- *Increase infrastructure accuracy*: since it becomes an automated operation, it will avoid the risk of manual configuration mistakes.
- *Apply Testing and Validation*: when a Policy is written via code, it is easier to test and validate it.
- *Create a Version control system*: since policies are managed via code, it is possible to store and manage them via a version control system.

Typically, Policies are written in high-level languages, and their application can be executed either via specific engine or integrating them with a new step of the CI/CD workflow.

In the case of the Policy Engine, it will receive data generated by the code as input, and will generate one of three possible results: a Warning, an Error or a Success. In the first case, only an alert will be generated to advertise the user about a minor problem, while in the second case an interruption of the execution will be performed.

3.4.1 Pulumi CrossGuard

Pulumi CrossGuard is a tool offered by Pulumi to integrate Policy as Code in the Infrastructure as Code approach.

It is a component of the Pulumi engine, and it is based on the three concepts of: Policy Pack, Policy and Enforcement level [24]. When the Pulumi engine receive the operation to execute, if the option *"--policy-pack"* is active, before their actual execution, the verification process will be performed.

A Policy Pack is the set of related Policy that will be validated on a Pulumi script. To create a new Policy Pack it is possible to use via command line interface the command "*pulumi policy new*" and as for the creation of Pulumi projects, it is possible to instantiate the project based either on some Pulumi examples or on custom projects. It can be written in Python, Javascript or Typescript and can be applied to any Pulumi stack written in any language. Inside a Policy Pack we can find three different sections: the import of all the dependencies, all the available Policies, and the definition of the actual Policy Pack object. This last component is composed by a name, and a list of policy names chosen from all those available that will be executed.

A Pulumi Policy is a block of code deployed inside a Policy pack and it aim to validate a specific rule that has been defined. A policy is composed by two sections: a function that will be executed when the policy has been invoked, and an Object that will be responsible of manage the specific Policy. This Object can be of two distinct types, *ResourceValidationPolicy* or *StackValidationPolicy.* The main difference is that the first will invoke the policy function for each resource of the Pulumi stack before this last has been registered, while the second will invoke the policy function only once, passing as argument all the stack after all the resources have been registered.

When the Policy Object invoke a validation function it will pass two arguments: the resource/stack arguments based on the typology of the policy, and a ReportViolation callback, that can be used to report the violation of the policy and return a violation message.

The Enforcement level is how the policy will act against a violation. It can be in three versions: mandatory, advisory or disabled. The first one will interrupt the code execution, and the entire stack will not be deployed. The second will only print warning messages, to advertise that a violation has been registered. The third one disables that policy. All these levels can be useful for defining different control over the infrastructure.

To run the Policy Pack in conjunction with the infrastructure deployment, it is necessary to add to the command "*pulumi up"* the option "*—policy-pack *path_to_the_package** *".*

3.5 Actual existing tools for FinOps implementation

Nowadays, there are more and more new solutions on the market that can be used by companies to implement FinOps practices. Before proceeding with the description of the developed processes, a comprehensive analysis on these tools can provide a complete overview of their method of operation and limitations.

The analysed tools are third parties' tools that companies can buy or pay-per-use, and are developed by: Aptio, an IBM company; CloudSaver a cost optimization provider; Infracost, a FinOps infrastructure application available for Terraform; CloudBolt, a cost management framework for Hybrid Cloud infrastructure.

3.5.1 Aptio Cloud financial management

The Aptio company, born in the 2007 and acquired by IBM in the 2019, is an organization that operates in the financial sector, offering management solutions for companies.

They offer three types of solutions that can be used by FinOps teams: a Technology Financial Management, a Cloud Cost management and an enterprise Agile Planning solution. All of them are more focused on the operational section of the FinOps team, indeed they don't supply and support for the development part.

The Cloud Cost management solution [25] is divided into two different options: a Cloud Financial management solution and a Cloud Total cost of Ownership solution. The first option focuses on the reduction of costs via the consolidation of multiple Cloud Provider's billing files into an easier platform where it is possible to set some Key Performance Index (KPIs) to understand where cost reduction can be performed. The second solution is an observability solution that can perform a mapping of all the services of multiple cloud providers to a standard taxonomy, helping in the rightsizing of the infrastructure to reduce cloud costs.

This solution can be useful for companies as it can help reduce cloud costs with easy integration into the business organization, however, its main limitations are its inability to provide proactive or automated tools for scheduling that are the main methods for proper and advanced FinOps practices.

3.5.2 CloudSaver Tag Management

CloudSaver is a company born in the 2018, that offers cost optimization solutions [26]. Their options are focused only on the AWS Cloud Provider, and the application of FinOps practices is based on Tag Management and a Map Manager that provides a more complete overview of all the deployed resources.

The tag Manager is an automated tool that can help companies in tag management. It is thought to provide a higher security and governance over all the resources, allowing the management of all the tags via a unified platform. The Map manager will provide a proactive control for the tag application since it will be possible to enable the auto-tagging option that will automatically apply tags based on some rules defined by the administrator.

The advantages of this solution are based on the proactive and automated approach that can help FinOps team, however, this solution present many limitations due to the possibility of application over a single cloud provider, the limited FinOps application to only tag management, and the lower integration with development service. Indeed, their solution are less compatible with Infrastructure as Code approach since they would bypass the code tag application and would not provide any additional service for coding.

3.5.3 Infracost FinOps tool

Infracost is an open-source project created in the 2020 which aims to create cost estimates for Terraform an IaC provider, via VsCode, CLI and external CI/CD systems [27].

Since the possibility to work with an IaC provider, Infracost is able to apply FinOps practices into a Multi-Cloud environment and also all the approaches are structured on a proactive methodology.

This tool is composed by three different components that deal with specific FinOps problems and are available starting from a restricted base free version to a pay-per-use complete option: the Infracost VsCode extension provides financial information about the resources that will be deployed; the CI/CD extension allows to add an analysis over the cost impact of the deployment; the Infracost Cloud tool allows to set some guardrails over budgets, policy control and report generation.

This solution can provide high support for FinOps practices inside companies, which an elevated grow margin, however it presents some limitations due to: the only possibility

to use Terraform as Infrastructure as Code provider, which is based on a Domain specific language which reduce the portability of the code; the lack of automation processes like tag automated application; the difficulties of integration with existing systems.

3.5.4 CloudBolt management tool

CloudBolt is a company born in the 2012 with the aim of provide a management system for Hybrid Cloud infrastructure [28]. Nowadays it includes different solutions from the Cloud Cost management to the automation processes for report generation.

CloudBolt offer a platform where it is possible to manage cloud costs, via the possibility to set internal quotas for the maximum number of resources or maximum costs that team can allocate, the possibility to set expiration date for unclaimed VMs and the possibility to automatically generate cloud cost reports.

Even if this solution started for Hybrid Cloud, today is also available for Multi-Cloud environment, however some limitations about FinOps practices of this solution are represent by the limited application of proactivity in code deployment, the lack of automation processes regarding tags management and reduced security over resources limit since they are constrained to the usage of this external tool to verify resource deployment.

3.5.5 Consideration over the existing tools

All the existing tools present advantages and limitations that can bring benefits to FinOps application in an organization. However, as can be evinced from the Table 3, each of them focuses on a specific particular aspect of the FinOps practices omitting other aspects that, due to the structure of the product, cannot be easily expanded in the future.

Moreover, many of them presents constraints over specific companies, for example CloudSaver with AWS or Infracost with Terraform, that limit the usage of these solutions and risk to limit also companies who want to adopt these tools to third parties' products.

Consequently, in this thesis work a more complete solution will be developed, which will represent a base and configurable set of processes that companies can perfectionate based on their specific need, providing the possibility to achieve a higher maturity in FinOps practices across a Multi-Cloud infrastructure.

	Aptio	CloudSaver	Infracost	CloudBolt
Works with IaC tools	No	No	Yes	No
Allows management operations	Yes	Limited for only tags	No	Yes
Allows proactivity and automation	Limited to report generation	Limited for only tags	Yes	Limited to budget or resources quotas
Works with multiple Cloud Providers	Yes	No	Yes	Yes
Can perform observability	Yes	No	No	No
Can be easily integrated with existing services	Yes	Yes	No	No

Table 3 - Comparison between the existing FinOps tools

4 Development of new FinOps processes

The development of new FinOps processes can vary over vast operations, which depend on the specific needs of the company that will have to apply them.

Many times, it is not possible to use existing generic solutions in their entirety. However, by following FinOps guidelines and principles it is possible to modify these solutions to fit the use case.

Following the analysis on innovative FinOps practices and based on the FinOps challenges previously cited, the development of new FinOps processes will be presented in this chapter. These processes will focus on proactivity and automation, creating a base solution which companies can extend for their use cases.

All these new processes will be created availing of tools as Pulumi, Jenkins, GitLab and Python scripts, that all together will form a complex solution that will provide added support for FinOps practices. The developed processes are three:

- A process to automatically apply and verify resources' tags into a CI/CD pipeline for the deployment of Multi-Cloud infrastructures.
- A process to execute periodically a script that generates a report about a typology of anomalies and can also execute a form of corrective actions.
- A process to manage and verify budgets over multiple Cloud Providers via a Pulumi Project.

All of the operation performed into these processes can be divided into three sections and this sub-process can be used either independently or as different parts of a single system.

After the presentation of the interactions between Users and the processes, and between Jenkins, Pulumi and the Python script, the three sections will be described: in the first section, proactivity will be achieved via the development of a Policy Pack, that can perform multiple controls before the actual deployment of a cloud infrastructure; in the second section the automatic generation of an anomalies and cost report will be explored via a Python script; in the last section automation and Continuous Integration/Continuous Deployment will be achieved via different Jenkins Pipelines.

4.1 The interaction models User-processes and Jenkins-Pulumi

The methodology of application of the FinOps practices depends on the specific needs and use cases of the company. However, to create a complete and working solution, an interaction model is needed.

In this way, all the processes will create a base solution that will be already available and also it will help in understand how them could be integrated with other functionalities.

The solution is based on four components:

- Pulumi: the IaC tool that will offer two different tools:
    - An IaC engine
    - A Policy validator
- Jenkins: the automation and CI/CD tool.
- Python scripts: that will perform multiple operations.
- GitLab: a repository where to store file and projects.

The interaction model can be divided into two parts: one that represents how users will interact with the system, and the other that represents how various parts of the system will interact each other.

In the first interaction, users could be the either developers or administrators, and both of them will interact with GitLab and Jenkins. With GitLab, developers will have to push all the Pulumi Project that they will create, while administrators will be able to modify the Policy Pack available on the repository and push user tag information files. With Jenkins both of them will be able to execute some Pipelines: developers can execute only the pipeline for the Project deployment, administrators can execute also other pipeline to manage users' information or Budgets and execute the automatic reporting process.

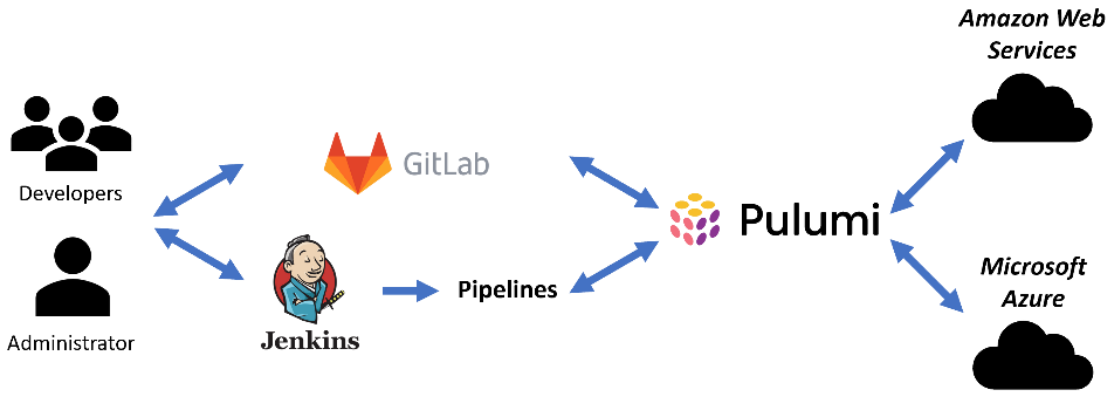


Figure 10 - Iteration Flow between Users, Jenkins, GitLab, Pulumi and the Cloud Providers

For a correct and automatic application of tags, administrator can create via a Jenkins pipeline a user tag information file, whose name will be formatted by the pipeline with the username of the user, and that will store all the correct tags that the user should use for his projects.

In the second interaction, Jenkins will interact with GitLab and either the Pulumi Engine or the Python scripts. Via the first one it will retrieve all the projects, user's tags information and the policy pack. In both Pulumi and Python cases, Jenkins, after having completed a list of operation of the pipeline, will perform the proper command to execute either the Pulumi script or the Python script.

With the application of this solution, users will not have enough permission to execute a Pulumi script without passing through Jenkins which will perform some previous operation like tag application and will also ensure the execution of the policy in the correct way. Furthermore, only Jenkins will have the credentials to deploy resources over the Cloud Providers reducing the possibly to outcome this methodology.

4.2 Pulumi Policy Pack

The Pulumi policy pack is a tool offered within Pulumi Crossguard, that can perform some previous verification on the Pulumi code developed before its deployment.

To achieve proactivity and to integrate FinOps practices, the Policy pack developed will offer three main characteristics:

- *organizational control, via Tag management*: each project will have the proper tag reducing the "mistagging" problem. This problem is the unintentional application

of a wrong tag to a resource and can be caused by different factors like the misspelling of the tag, or the application of a tag from another project. This verification, together with the tag automatic application, will help FinOps teams in the generation of cost reports, in the detection of cost anomalies, and in the management of multi-cloud resources.

- *financial control, via actual and forecasted Budgets verification*: the control of actual and forecasted spend can be difficult to be managed in big companies, and it is easy to overcome them. With this verification, if the policy has been violated, a corrective action will be required before the generation of new costs.
- *deployment control, via the resource's Region deployment*: due to the Cloud Service Provider infrastructure, each deploy Region can offer different prices for the same service and for some companies, some Regions may be more cost-effective than others. With this control each company will select proactively all the available regions over multiple Cloud Providers, and the complete Multi-Cloud infrastructure will not be deployed in case of policy violation.

The typology of policy and the enforcement level will vary between policies, and this last will be editable among the three options available.

A Policy Pack will be able to work over Multi-Cloud project and it has to be defined for at least the specific Cloud Providers whose resources it will validate. In this thesis work, the Policy Pack developed will supply policies available for Amazon Web Services and Microsoft Azure.

Each Policy have to be written so that it can work for every Cloud Provider defined, since is not possible to define at priori which policy should be run against a specific resource. This is due to the formatting of the resource properties that can differ form resources of a Cloud Provider to the other even if both are deployed in Pulumi. However, each policy will be able to detect the typology of resource and its Cloud Provider, to be able to perform the correct verification.

4.2.1 Region Policy

The first policy developed is the Region control policy named *"region-lock"*. It aims to verify in which region of the Cloud Provider, the resource will be deployed.

In this Policy, the administrator will define a list of regions via their identification name, for example for AWS there could be "*us-east-1"* for the region in US East in Virgina or *"eu-south-1"* for the region in Milan, while for Azure there could be *"eastus"* for the region in the US East, and *"francecentral"* for the region in France.

The policy will invoke the function "*region_enforce_validator*" and since it is a Stack Validation Policy, it will be executed only once. The control will be performed on each resource after it has been registered in the stack and this will avoid the repetition of the policy multiple times. Its enforcement level is set by default to "*Mandatory"* since the company should deny any occurrence of deploy resource outside the available lists.

4.2.2 Budget Policies

To verify budget violation two different policies have been developed: one to validate the actual spent, and the other to validate the forecasted spent. Both of them are required to offer a complete verification over a defined budget, and the modality of reporting the violation can vary.

Before the description of the policies, some considerations have to be taken.

Via the automation Jenkins Pipeline, the FinOps team will be able to create either complete budgets or per project budgets both divided into the two Cloud Providers. In the first case, a generic name can be assigned to the Budget, while in the second case, the budget name will depend on the project name.

Since even a unified management process over the budgets exists for the execution of the policies, both budget on the different Cloud Providers will be checked. To do so there could be two different options:

- A single Policy that verifies both the Cloud Providers budgets
- Two separated Policy that validates each a single Cloud Provider

For the development of these policies, the second option has been selected since it allows to verify both Cloud Providers during the same execution, while in the first situation if one budget has been violated, the other would not be checked.

In addition, for the development of these policies the external tool Boto3, and some REST APIs will be used instead of Pulumi AWS and Azure APIs.

During the execution of a Policy Pack, Pulumi does not allow to instantiate other resources and it was not possible to use the Pulumi AWS and Pulumi Azure APIs since all of them in any case would create new resources. To outcome this problem, for AWS the Boto3 SDK will be used, that is an AWS SDK for Python that can invoke AWS APIs, while for Azure will be invoked the REST APIs for the budget management.

The first typology of policies are the Actual Budget policies, and they verify if the budget limit has been exceeded.

Both the AWS version named *"total_budget_aws_policy"* and the Azure version named "*total_budget_azure_policy*" are *StackValidationPolicy* since they have to be executed once for all the stack.

The first will invoke the function *"check_budget_aws"* and this validation function will retrieve all the budgets of the company's account and control if the required budget exists. Once the budget has been identified, the information about the actual spend and the budget limits will be extracted and compared.

The latter will invoke the function *"check_budget_azure"* and this validation function will firstly invoke a third function named *"get_azure_budget"* to retrieve a specific named azure budget and after will perform the verification over the actual spent and the budget limit.

The function *"get_azure_budget"* is external to the validation function since it will be also used from the forecasted version. It will invoke a REST API via a Microsoft azure endpoint, and after will parse the response to a JSON format.

The second typology of polices are the Forecasted Budget policies and they verify if the spend prevision for the end of the month has been exceeded. This type of policy is less restrictive in comparison with the previous verification since its Enforcement level has been set to "Advisory" to avoid the interruption of the deploy. However, it could help FinOps teams to apply corrective actions to the project or some budget modifications before the budget has been exceeded and also without having the deploy interrupted.

Both the AWS version named *"forecasted_budget_aws_policy"* and the Azure version named *"forecasted_budget_azure_policy"* behave as the actual spend versions, however the main difference is in the filed of the budget that they will verify: after the two

verification functions *"check_forecastedSpend_aws"* and *"check_forecastedSpend_azure"* have retrieved the budget information they will compare the budget limit with the forecastedSpend field.

In all the policies if the actual spend or the forecasted spend exceed the budget limit, a report violation will be generated via the ReportViolation callback.

4.2.3 Tag Policy

Resource tagging represents an important practice for FinOps operations inside a company. It can ensure an easier and more correct way to manage resources, allowing to find, group and control them.

The aim of this policy is to guarantee the correct application of tags for all the deployed resources in all the Cloud Providers.

In order for this solution to work properly, the FinOps teams should before select a correct set of tags, and assign them to users: in this thesis work a GitLab repository will contains a set of file, one for each user that could use the process, whose name will be formatted by the username of the user, and that will contain the set of tags for that user.

Before the execution of the Policy, as soon as the policy pack has been executed, the user tags information file will be imported and converted into a Python dictionary.

Both the version of *ResourceValidationPolicy* and *StackValidationPolicy* have been developed: in the first case the violation will be reported for all the resources, while in the second case as a resource triggers the violation all the process will be stopped. Beside this difference, in the code execution there are no variation since for the stack version there will be an iteration among all the resources. Furthermore, both the policies will work for both AWS and Azure, since the verification will be executed on all the resources, and a separated version of them would perform unnecessary computation.

The two policies are named *"resource_tag_enforce"* and *"stack_tag_enforce"* and respectively will invoke the validation functions *"resource_tag_enforce_validation"* and *"stack_tags_enforce_validation"*. The validation functions will verify on which Cloud Provider the resource will be deployed to extract the correct field about the resource tag since the resource representation can vary from one to the other Cloud Provider. After it

will match all the applied tag to the resource with the required tags imported from the user information.

With this method, users will not be able to modify without permission any tag to the resource, and also the mistagging problem will be avoided.

4.3 The automated reporting script

Report generation and anomalies detection are some of the principle FinOps challenges which FinOps team mostly incur. Without a complete strategy, the identification of anomalies and a good methodology of reporting them could be insidious.

Even if many Cloud Providers offer different tools for anomalies detection, none of them can be automated and supply a complete report. One typology of anomaly that is also compliant with the other developed process, is represented by the untagged resources, within or outside allowed regions that can generate some costs.

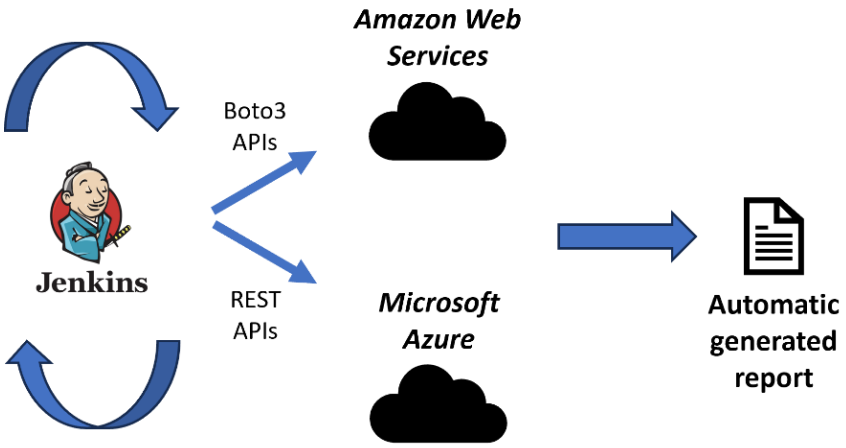


Figure 11 - Automated reporting generation process

This process aims to generate a complete report over the Multi-Cloud infrastructure, reporting all the resources deployed outside the list of allowed regions without been tagged and that could generate costs. This type of anomaly represents a higher difficulty in its management, as it is more complicated to trace. Many Cloud Providers allows to identify tagged resources that generate costs, however if the resource is untagged there is no way to identify it.

The execution of this process can be divided into two parts: one is represented by the Jenkins Pipeline which automate the report process by repeating the execution of the

script after a period of time defined by the administrator and also send via mail and pushes on the GitLab repository the generated report; the second part is represented by the actual script which performs different operation to locate the resources, and following generate a markdown report with also the possibility to perform some operation on the obtained resources.

The script is written in Python and can be divided in three sections: the first will define and declare the operation via Boto3, the AWS SDK for Python; the second will define them via REST APIs for Azure; the third will execute all the functions and generate the report.

In the AWS part, primarily will be declared a Boto3 client account variable, which will be used to retrieve all the other components, and other two variables: the "*regions"* variable which is a list obtained by the account variable, and an empty dictionary *"resource_no_tag"*. After, four functions will be defined:

- *get_aws_untagged_resources*: this function will iterate over all the regions, adding them to the empty dictionary, and then will retrieve all the resources grouped by regions filtering them by the tag field.
- *get_aws_cost_resources*: this function will use the Boto3 CostExplorer to retrieve all the resources that in the last 14 days, which represent the maximum time-shift available, generated costs. However, due to an AWS limit that will be improved, this function can retrieve only EC2 instances [29].
- *evaluate_untagged_costs_resources:* this function will invoke the two previous functions to retrieve all the information and after will execute the intersection finding all the resources that are not tagged and that generate costs.
- *tag_untagged_resources:* this function will apply some special tags to the untagged resources, so that FinOps team can manage them easily. It may or may not be executed depending on the choice the administrator defines during the application of the pipeline.

In the Azure part, the function structure is different, since the resource organization of the Cloud Provider is different. For this part of the report will not be identified all the untagged resources but the one that belong to untagged resource group. If a resource

belongs to a resource group that has been tagged il will be still easily identified, while if also the resource group is untagged, it becomes more complicated.

Since all the operation will be managed via REST APIs in this part there is no necessity to declare in advance some global variables.

The developed functions are four:

- *azure_token*: this function will retrieve some environment variables that represent the credentials for the Azure Cloud Provider to obtain via a POST request a Oauth2 token.
- *get_azure_untagged_resource_groups:* this function will accept one parameter that will be the Azure token and it will perform a GET request to retrieve all the resource groups available for a specific subscription. In Azure a subscription represents a logical unit of services liked to an account [30]. After, all the resource groups will be filtered to obtain the untagged ones.
- *get_azure_untagged_resources*: this function will accept two different parameters, the Azure token and a resource group name. It will perform a GET request to retrieve all the resources in a resource group, and then it will filter them to find the untagged resources.
- *get_azure_cost_resources:* this function will accept two parameters, the azure token and a resource group object. It will perform a POST request to obtain the resources that in the last 14 days have generated costs sending a query written in the request body. Subsequently it will filter the response obtaining the untagged resources that have generated costs. With Azure there is not the limit to 14 days for the request, however, to be compliant with the AWS sections it has been limited to this value.

In the last section of the script, the generation of the report is performed via the file management available in Python. Also, the previous function will be invoked to obtain all the data needed for the report generation.

A further control is executed over the number of parameters that have been passed at the execution of the script, since if a parameter has been passed, then the untagged resource will be tagged.

REPORT

This report contains the information about AWS untagged resources that are deployed in regions outside the default one or that have a costs and Azure untagged resources that are deployed in untagged resource groups, the list of all the untagged resources groups and the untagged resources that have a cost

AWS resources

AWS untagged resources:

eu-south-1:

ResourceARN
arn:aws:ResourceType:RegionCode:Subscription:Resource
arn:aws:ec2:eu-south-1:xxxxxxxxxxxx:dhcp-options
arn:aws:resource-groups:eu-south-1:xxxxxxxxxxxx:group/named-group

eu-west-1:

ResourceARN
arn:aws:ec2:eu-west-1:xxxxxxxxxxxx:security-group/sg-xxxxxxxxxxx
arn:aws:events:eu-west-1:xxxxxxxxxxxx:rule/AutoScalingManagedRule

AWS untagged resources with a cost in the last 14 days:

none

Figure 12 - First part of a report sample containing the introduction and the AWS section

AZURE resources:

AZURE untagged resource groups:

Resource Group id	Resource Group name	Resource Group location
/subscriptions/aaaaxxxx-aaxx-aaxx-aaxx-aaaaxxxx/resourceGroups/DefaultResourceGroup	DefaultResourceGroup	francecentral
Resources without tags:		
None		
/subscriptions/aaaaxxxx-aaxx-aaxx-aaxx-aaaaxxxx/resourceGroups/resourceGroup_name	Resource Group name 1	francecentral
Resources without tags:		
resource_name1		
resource_name2		
resource_name3		
/subscriptions/aaaaxxxx-aaxx-aaxx-aaxx-aaaaxxxx/resourceGroups/ResourceGroupName2	Resource Group Name 2	northeurope
Resources without tags:		
resource_name1		
resource_name2		
resource_name3		
resource_name4		

AZURE untagged resources with a cost in the last 14 days:

Resource id	Resource Group name	Resource Cost
/subscriptions/aaaaxxxx-aaxx-aaxx-aaxx-aaaaxxxx/resourcegroups/ResourceGroupName/providers/microsoft.storage/storageaccounts/StorageName	storage name	0.702749967
/subscriptions/aaaaxxxx-aaxx-aaxx-aaxx-aaaaxxxx/resourcegroups/ResourceGroupName/providers/microsoft.storage/storageaccounts/StorageName	Storage name	0.0419266286
/subscriptions/aaaaxxxx-aaxx-aaxx-aaxx-aaaaxxxx/resourcegroups/ResourceGroupName/providers/microsoft.web/sites/webSiteName	Website name	2.5727162836

Figure 13 - Second part of a Report sample containing the Azure section

In the Figure 12 and Figure 13, a sample report is presented, divided into the two different sections for AWS and Azure. In the AWS section it also possible to notice what happen in case of no untagged resources that generate costs, while in the Azure part three different examples are provided. Moreover, in the AWS part all the resources are grouped together, while in the Azure part they are grouped for untagged resource group, and there could also be untagged resource groups without any untagged resource. Where are present the

letter “a” and “x”, it means that sensitive information are present, and the “x” means that there are just numbers, while “ax” that could be either letter or number. In both AWS and Azure cases, they represent the code of the account or subscription that has been used.

4.4 Jenkins Pipelines

Another important characteristic to increase the quality of FinOps practices and achieve a higher maturity level, is the automation of the processes. Jenkins, as an automation and CI/CD tool, will represent in this thesis work the interaction tool where developers and administrators will only need to execute the pipelines to obtain the desired results.

Each Jenkins Pipelines will represent a different FinOps process with a specific purpose, and it will interact with the previous cited components to perform different tasks.

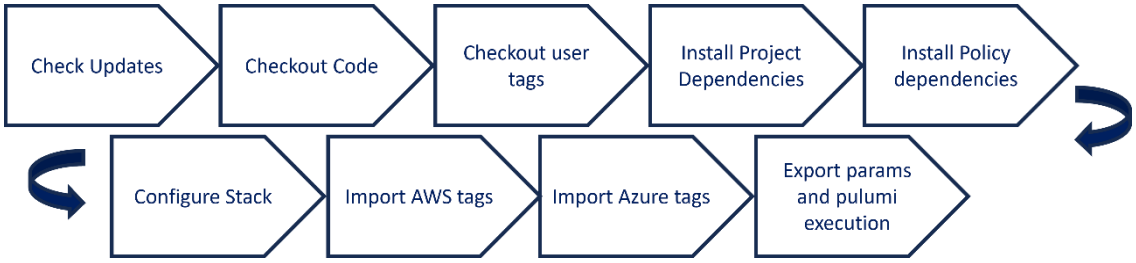
4.4.1 AutoTaggablePipeline

The first pipeline developed is the “*AutoTaggablePipeline”,* and it represent the main process for the deploy of a multi-cloud infrastructure. It aims to perform some operation before the deploy of a cloud infrastructure without any supervision of the developer. It represents the part of Continuous Deployment of the CI/CD process since after the developers will have written, built, tested the code it will perform in an automatic way the deployment of the infrastructure.

It is a parametrized pipeline, so it will accept some parameters before the execution. In this case, two parameters are required:

- *The Pulumi Project name*: a string parameter so Jenkins will be able to detect the correct project to execute.
- *The Pulumi Stack name*: a string parameter that identifies the Pulumi Stack where the project will be deployed.

It is composed by nine different stages as represented in the Workflow 1 and each of them will perform a task that in case of failure will terminate the pipeline execution. In order, the stages are: *“Check updates”, “Checkout code”, “Checkout user tags”, “Install Project dependencies”, “Install Policy dependencies”, “Configure Stack”, “Import aws tag”, “Import Azure tags”, “Export params and pulumi execution”.*



Workflow 1 - Representation of the workflow of the AutoTaggablePipeline

In the first six stages, the pipeline will: control if Pulumi and Python are installed; checkout the Pulumi Project and the User tags from a repository detecting the correct file via the Jenkins username of the developer who is executing the pipeline (i.e. GitLab for this thesis work); install all the dependencies for both the project and the Policy code; select for the Pulumi Project the correct Stack whose name has been passed by the user as parameter.

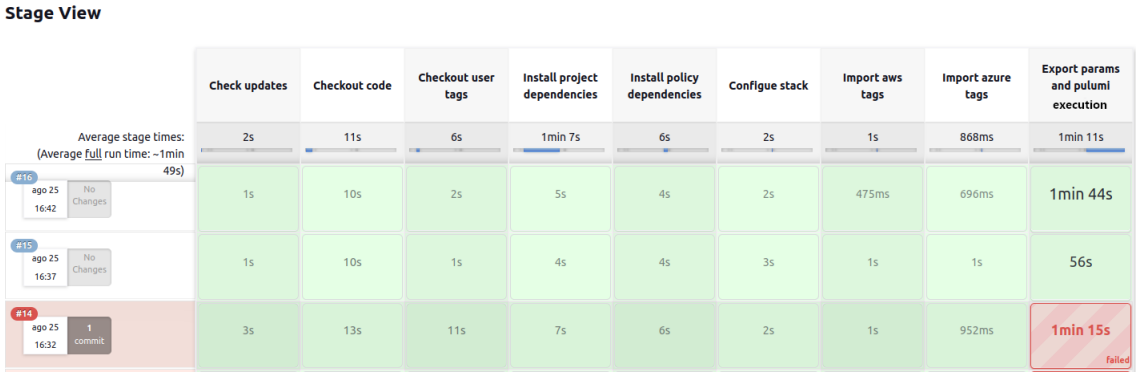


Figure 14 - AutoTaggablePipeline Stage view with 2 positive executions and 1 failure

After them, some core stages will be executed. To automatize the resource tag operation two different approaches have been applicated:

- For AWS, a modification of the Pulumi Stack configuration file will be executed since for its configuration the section “aws:defaultTags” was available and when Pulumi deploys the resources if possible, it will apply this configuration.

```
1. config:
2.   aws:region: eu-south-1
3.   aws:defaultTags:
4.     tags:
5.       Project: Tesi
6.   azure-native:location: francecentral
```

Code 2 - Pulumi YAML Stack configuration file modified

- For Azure there was no configuration sections available for the tag application, for this reason an alternative way will be used. Jenkins will add to the Pulumi Project a block of Python code described in Code 3 that is composed by two functions and an execution command. The two functions will be a Pulumi Resource transformation functions [31], which is a special option that can override some components of the resources before they will be deployed. The override will consist in the identification of the resource to understand if it is taggable, and then in the modification of the resource properties adding the tags. Since almost all the Azure resources have to be deployed into a resource group and so are easily identifiable, in this case only the resource group will be tagged.

```
1. def is_taggable(type_: str) -> bool:
2.     taggable_types = ['azure-native:resources:ResourceGroup']
3.     return type_ in taggable_types
4.
5. def register_auto_tags(tags: dict):
6.     def auto_tag(args: pulumi.ResourceTransformationArgs):
7.         if is_taggable(args.type_):
8.             if args.props is None:
9.                 props = {}
10.                new_tags = tags
11.            elif args.props['tags'] is None:
12.                props = args.props
13.                new_tags = tags
14.            else:
15.                props = args.props
16.                new_tags = {**props.get('tags', {}), **tags}
17.            props['tags'] = new_tags
18.            return pulumi.ResourceTransformationResult(props, args.opts)
19.     pulumi.runtime.register_stack_transformation(auto_tag)
20.
21.
22. required_tags = yaml.load(open('../tags.yaml'),Loader=yaml.FullLoader)
23.
24. register_auto_tags(required_tags)
```

Code 3 - Azure automation tag script written in Python

The last stage that will be executed is the export of all the necessary environment variables and then the execution of the command "*pulumi up --policy-pack ../policyPack*", which will execute the deploy of the infrastructure only after the policy have been validated.

In this way, the developer will neither need to apply the tags to the resources and apply the policy pack before the deployment.

4.4.2 UserTags

This Pipeline is a management pipeline, and the administrator will be able to create, a new user tag information file that will be correctly formatted and named to makes all the other processes works.

It is a parametrized pipeline and to be executed will need three mandatory parameters and a variable number of parameters defined by the administrator representing the tags that will be applied to the user. All these parameters are String Parameter and the mandatory ones are: username, email address and id of the user which the tags will be associated. The other parameters that represent the tags can be set by the administrator in different modalities since the pipeline is highly configurable.

It is composed of five stages: *"Check params", "Git Checkout", "Check if file already exists", "Create file", "Commit file"*.

Workflow 2 - Representation of the workflow of the UserTags pipeline

In the first stage, many checks are performed over all the passed parameters to avoid the possibility of mistake in the association process. After, the file will be created as a YAML file whose name will depend on the provided username.

4.4.3 HandleBudgetPipeline and ProjectBudgetPipeline

Cloud Providers offers tool for the management of different budgets. However, for Multi-Cloud projects or infrastructures there are no native tools to handle unified budgets correctly. In addition, a proper budget management represent one of the best ways to improve the company's FinOps team maturity in two different FinOps challenges: accurate forecasting spend, and Multi-Cloud cost reporting.

To handle this problem, two complementary pipelines have been developed: the *"HandleBudgetPipeline"* and *"ProjectBudgetPipeline"*.

These Pipelines will be used by the FinOps team or the administrator, since are management pipelines. Both of them will focus on the creation, update, and cancellation of a budget, which will operate over the two Cloud Provider adopted in this thesis work.

For a Multi-Cloud budget management many options could be applicated depending on the way of performing the operations: it can be possible to manage everything via an iterative script that Jenkins will execute repetitively after a period of time, via a report generation of various costs, or via the management of the existing Cloud Provider native tools. Since this solution will represent a base solution that companies can adapt for their use cases, for this thesis work the last option has been explored. It can perform a better integration with existing company system and also can be more reactive for alert generation with respect to the other solutions.

For each budget created via this process, two different budgets will be created in both Cloud Providers with a budget limit division set by the administrator. Each of them will have the alert activated, so in case one or the other will be exceeded a notification will be sent. However, even if two separated budgets will be created, both will be managed via the same Pulumi Project and will belong to the same stack. This will make the operation on them specular, and so they could be managed as a single entity.

The main difference between the two pipelines is that the first will manage a complete budget for the company's account that will cover all the cloud spent of that account, while the second will cover specific per project budgets considering in account only the project's resources.

Both *"HandleBudgetPipeline"* and *"ProjectBudgetPipeline"* are parametrized pipelines and are composed by six parameters the first and seven parameters the second. Also, both are composed by five stages. However, not all parameters are mandatory, since the same pipeline can perform three different operations: creation, update, cancellation of the budget.

Starting from the HandleBudgetPipeline, the mandatory parameters for all the operations are:

- *BudgetName:* A string parameter that will be used to assign the same budget name for both AWS and Azure.
- *Operation:* A choice parameter that can set the typology of operation performed by the pipeline. The available options are Create, Update, Delete.
- *Pulumi_Stack:* A string parameter that will set in the Pulumi Project the correct stack where the budget resources will be deployed.

In case the option selected is *Creat*e, the following must be added to the mandatory parameters:

- *BudgetLimit*: A String parameter that represents the total budget limit over both the Cloud Providers that will be set. This value will have to be a numerical value that is grater that 0.
- *SplitRatio*: A String Parameter that represents the modality of division of the budget over the different Cloud Providers. This value will have to be a numerical value included between 0 and 100, where 0 represent a complete budget in the first Cloud Provider and 100 a complete budget in the second Cloud Provider. In this scenario, the division will be performed considering as first Cloud Provider AWS and as second Azure, and the budget division will follow the following equations:

$$CP_{budget} = BudgetLimit * (SplitRatio)/100$$

Equation 1 - first Cloud Provider budget definition

$$CP_{budget} = BudgetLimit * (1 - SplitRatio)/100$$

Equation 2 - second Cloud Provider budget definition

In Jenkins there was not the possibility to include a numerical parameter, however in the first stage of the pipeline, further control will be performed.

In case the option selected is *Update*, instead of the *BudgetLimit* and the *SplitRatio* another parameter is required:

- *Percentage_variation*: A String parameter that defines a percentage of variation over the actual values. This value will be a numerical value but not limited, since if the budget goes under the 0, and error will be raised from the Cloud Providers.

For the options *Delete* no additional parameters are required.

In the second pipeline, *ProjectBudgetPipeline*, almost all parameter coincides with the previous pipeline. Only one of the previous parameters will be replaced with two new parameters that will help in the association of the budget to a specific project. The *Budget_name* parameter will be replaced with this key-value couple parameters:

- *Project_tag_name:* A string parameter that defines the key value of the tags.

- *project_tag_value:* A string parameter that defines the value of the tags. Moreover, from this parameter a suitable name will be generated for both the Cloud Providers budgets.

These two values will be used as filter tags for the budget's resources, in this way the budget will filter all the resources that owns these tags.

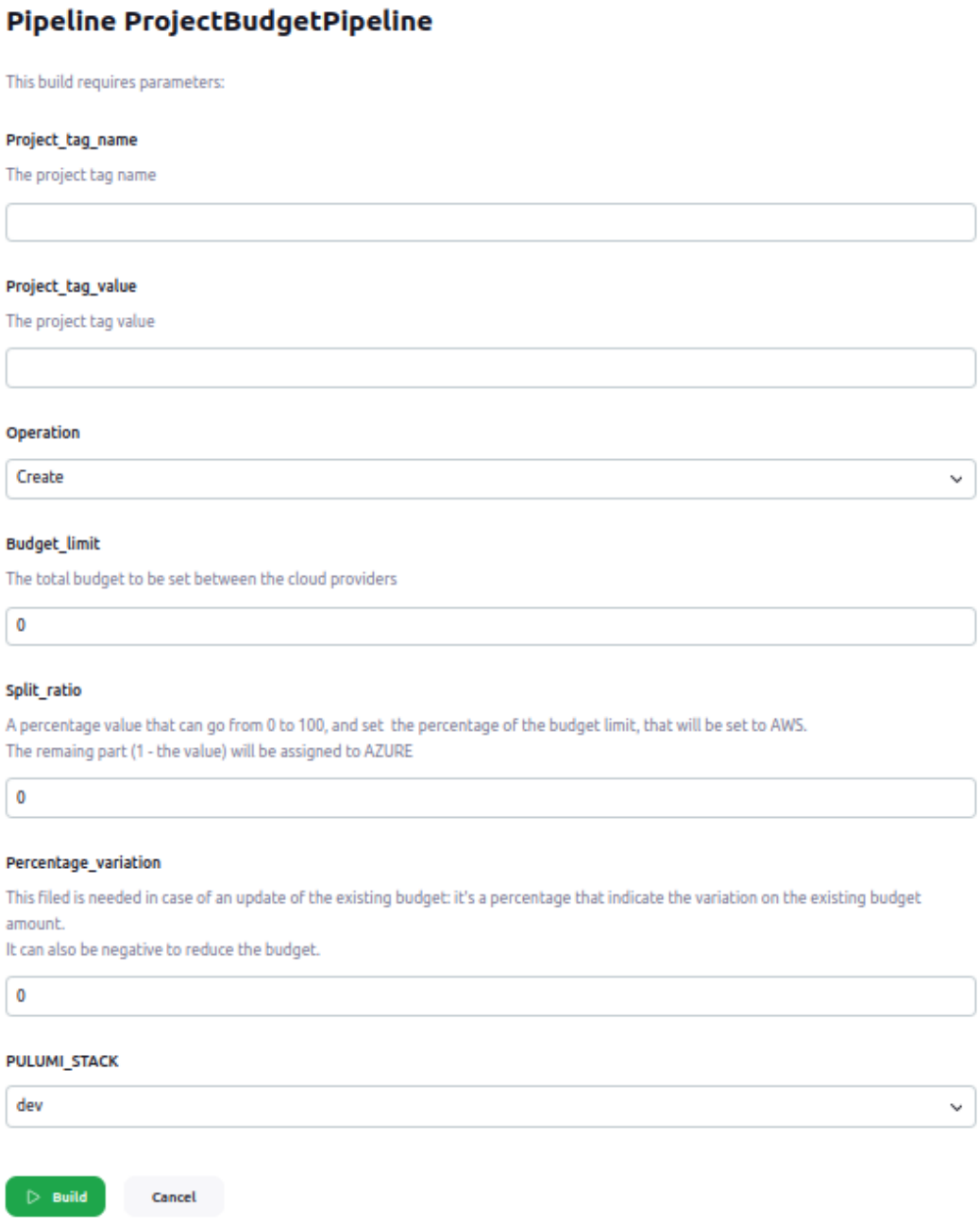


Figure 15 - ProjectBudgetPipeline build with parameters form

Both pipelines will execute the same stages that in order are: *"Check parameters", "Git checkout", "Install dependencies", "Configure stack", "Perform the operation"*.

Workflow 3 - Representation of the workflow of both HandleBudgetPipeline and ProjectBudgetPipeline

In the first stage, all parameters will be controlled, ensuring that are in the correct form, for example that the numerical parameter are not textual. Then, the Pulumi Project to manage the budget will be checked out from a GitLab repository, all the dependencies will be installed, and the Pulumi stack will be configured properly. In the last stage, the execution of the Pulumi Project will be performed without any policy control since it a well-defined project that does not depend on developers.

The Pulumi Project previously cited, will contains all the operation to manage the AWS and Azure budgets. In case of the per Project pipeline, the first operation performed is the generation of the budget name starting from the *Project_tag_value*, since not all the characters can be used in the Cloud Providers budget name. Otherwise, it will firstly control if the budget that will be defined already exists or not and if this is coherent with the typology of operation that will be performed (i.e., is not possible to create a budget that already exists or is not possible to update a non existing budget). After it will perform all the operation to assign the correct values to the budget, with the correct limit division.

4.4.4 PeriodicController

The last Pipeline developed will perform the control of the periodic verification over the complete Multi-Cloud infrastructure and the report generation. This Pipeline is named *"PeriodicController"* and it aims to execute the verification via the Python script, automate the report generation, alert the FinOps team or the administrator about the result of the report.

Also, it will push the report in a GitLab repository, to obtain a versioning system over this file.

This Pipeline is a parametrized pipeline composed by only one parameter, and it is composed of five stages and one post activity. In Jenkins a post activity is a set of execution command that will execute after the completion of the stages. It could be of

various typology: always, changed, fixed, regression, aborted, failure, success, unstable, unsuccessful and cleanup [32]. For this pipeline purpose the always modality represent the better choice since in this section the pipeline will send via mail the report generated and so even in case of pipeline problems the generation of this mail still be an alert for the receiver. Moreover, this pipeline has also a build trigger, which allows the repetition of the pipeline after a period of time in a complete automated manner. In this way, the complete automation of this system will allow FinOps team to receive periodically reports without any action.

The parameter required for the execution is:

- *"DefalutTags":* It is a checkbox parameter, which allows to activate the corrective actions of the Python script.

The five stages of the pipeline are: *"Check updates", "Checkout code", "Install project dependencies", "Execute script", "Commit file".*

Workflow 4 - Representation of the workflow of the PeriodicController pipeline

In the first stage only the Python installation will be checked. After, always from GitLab, the Python script to be executed will be retrieved and installed all the dependencies. The execution script will only invoke via a BASH command the Python execution and in case the checkbox parameter has been checked it will add a command argument that will activate in the script the execution of corrective actions.

In this way, the Python script can be modified adding new corrective actions without the necessity of modification for this pipeline.

5 Processes validation and testing

In this chapter, an experimental validation will be carried out to test the developed processes for the execution of the FinOps practices.

This analysis will be conducted via specific applications of the processes to verify their individual performance. Firstly, three different validations will be conducted over the three main processes developed: tag application, region deployment verification, budget management and forecasted anomalies detection. Consequently, a performance evaluation of the deployment pipeline and a comparison between the complete developed process and the existing tools will be performed.

5.1 Automation tag validation

Tag application represent one of the major FinOps challenges since via its application many other operations can be performed, like enabling automation processes, identifying unused resources, reporting on Multi-Cloud resource allocation, grouping Multi-Cloud project resources.

However, even if the assurance of the application of the tags has been performed via the Policy Pack verification, the automation process has to be verified and validated.

According to a statistical survey conducted by the CloudSaver company [33], dividing the respondent companies based on their annual cloud spent (ACS), as represented in Table 4, it was evinced that: for small size companies whose ACS was lower of 1 million dollars the untagged resources percentage was of the 17%, for medium size companies whose ACS was in the range of 1 million and 10 millions dollars, the untagged resources percentage was of 95%, for large size companies whose ACS was over 10 millions dollars, the untagged resources percentage was of 44%, with the overall average of the 52% of untagged resources among all the companies.

The gap between all this values can be explained considering the typology of companies: small companies are able to perform a greater tag application since they have to manage a reduced number of resources and the application of FinOps practices, that represent an innovative change in a company structure, can be carried out in an easier way with respect to medium and large companies; indeed, large companies have more difficulties in the tag

management, however their interest in FinOps practices and their possibility to invest in them are greater that medium companies possibility.

	Actual untagged resources percentage		
Company size:	Small	Medium	Big
Untagged percentage:	17%	95%	44%

Table 4 - Companies untagged resources percentage

With the application of the developed solution, the automation of tag application can guarantee a higher percentage of tagged resources. However, this percentage will depend over the taggability of the cloud provider's resources, since not all of them are enabled to be tagged.

In this validation, as it is possible to notice in the Table 5, a first theoretical result has been evaluated about the percentage of taggable resources among the two cloud providers AWS and Azure. Consequently, an experiment over a reduced set of resources have been performed to verify the correct application of the tag with the process and their taggability.

For the theoretical computation, via the list of resources of both Cloud Providers [34] [35] the percentage of taggable resources has been computed evaluating the number of taggable resources over the total number of them. To execute this evaluation, both the lists have been exported in a spreadsheet editor and for AWS, the evaluation has been computed by considering all the resources that had the field "Tag Editor Tagging" with "Yes" over the total number of resources, while for Azure, the evaluation has been computed considering all the resources that had the field "Support Tag" with "Yes" and that could not be instantiated in a resource group over the total number of resources. The distinction applied for Azure, has been done since if a resource belongs to a resource group and this last has been tagged, it will be still identifiable. For the AWS provider, the percentage of untaggable resources was of the 13%, while for the Azure provider, the percentage was of the 5%, with an average percentage of the 9% of untaggability.

For the experimental computation a reduced set of 45 different resources has been randomly selected out of 317 total resources available on Pulumi, so that the value would be as generic as possible. In this scenario the 11,11% of untagged resources (so 5 over

45) has been obtained via the deployment of the 45 different resources and a manual verification of the tag application. However, it should be considered that in case of an improvement of the taggability of the Cloud Providers, also the developed solution will present better results.

	Percentage of untagged resources (included untaggable resources) with the developed process			
Cloud typology application	Only AWS infrastructure	Only Azure infrastructure	Theoretical Multi-cloud infrastructure	Experimental Multi-cloud infrastructure
Untagged percentage	13%	5%	9%	11,11%

Table 5 - Percentage of untagged resources with the deployed solution

Moreover, a comparison between all the data presented in the previous cited research and in the theoretical and experimental analysis, can be carried out in the Figure 16, analysing per company's dimension the variation of the application of the developed processes in different scenarios.

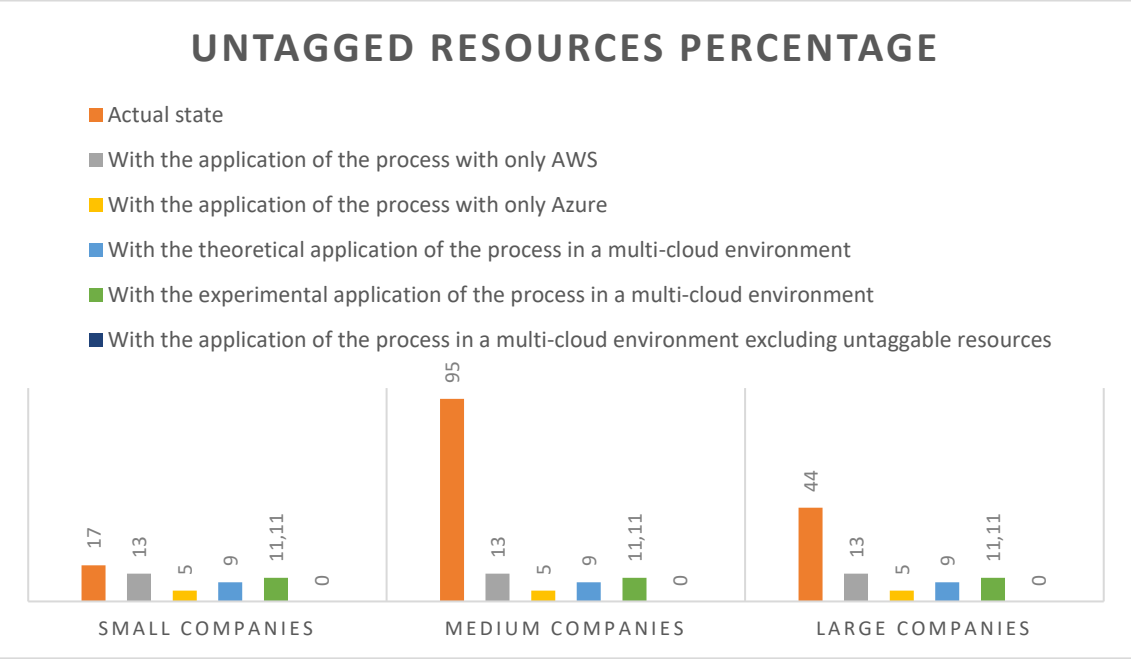


Figure 16 - untagged resource percentage per companies' dimension

As can be noticed, the theoretical evaluation and the experimental taggability percentages are comparable since each resource weights a variation of 2.22 points percentage, and the difference of 2,11 points between them can be caused by the reduced set of resources for the test.

Moreover, in all the scenarios a reduction of the untagged resources can be obtained, with greater benefits for medium and large companies which performs higher difficulties in tag management and application. However, it must be taken into account that a complete application of tags will depends on the possibility of tag application supported by the Cloud Providers.

5.1.1 Mistagging resolution

A further consideration about the advantages of the automated tags application can be carried out analysing the mistagging problem: many times, if tags are applicated manually, users can incur in grammatical errors both in the key or the value of the tag.

Since a tag is composed by two parts, the chance of getting either one of them or both wrong is higher. For example, considering the tag "Project: Application", the key could be written as “project” with the lowercase initial, or the value could be written “application” with the lowercase initial, or “app” abbreviate, or misspelled as “aplication”.

As reported in the CloudSaver survey [33], always dividing the companies for their dimension, the number of tag applicated only for the typology of AWS EC2 and AWS EBS resources are: for small companies of 37 for the first type and 20 for the second; for medium companies 135 for EC2 resources and 59 for the EBS resources; for large companies of 216 tags for the first typology and 159 for the second. These values of tags are highly grater than estimates where core tags for each typology of resource should be of 3-7 tags plus a variable number of tags for each project.

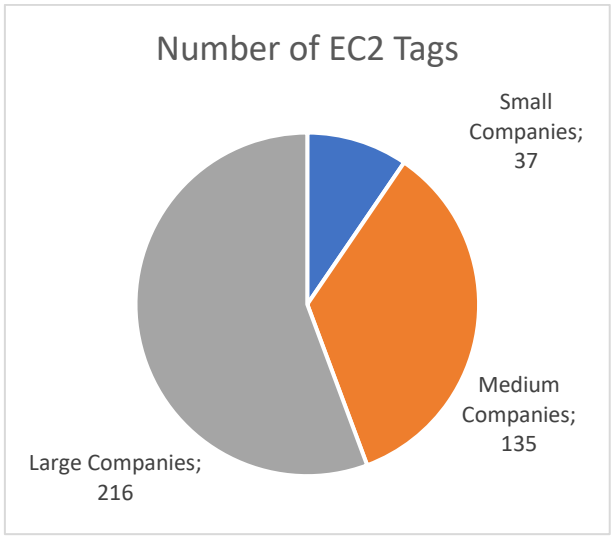


Figure 17 -CloudSaver analysis over the companies' number of EC2 tags

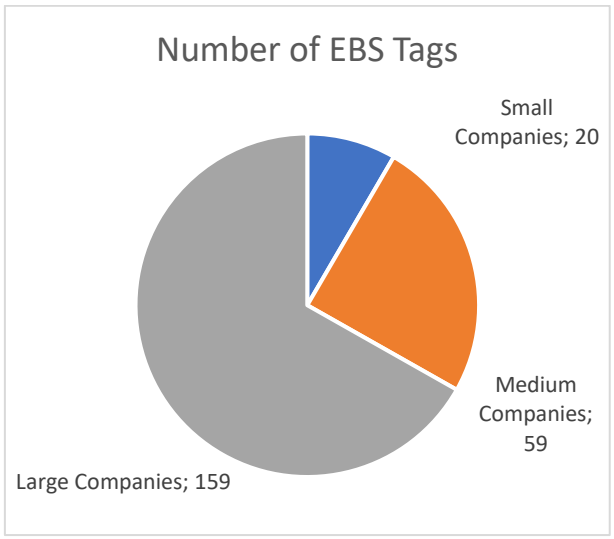


Figure 18 - CloudSaver analysis over the companies' number of EBS tags

With the developed process, also this problem can be avoided since the definition of the tag will be performed only once by the administrator and consequently, it excludes any manual tag application, guaranteeing the correct key-value pair for each resource.

For example, it was possible to be seen from the 40 tagged resources deployed previously that the same correct tag was applied. However, it was not possible to verify on the same resources how many erroneous tags would have been applied with a manual process, since it would require the application of the solution in a real scenario.

5.2 Region policy application analysis

The selection of the deployment region can represent a cause of increased costs, since different regions of the same cloud provider can offer different prices for the same resources. Moreover, in the presence of saving plan, the usage of different regions instead of the default one, can increase this problem even further.

Some Cloud Providers, already allows to perform some operation over the deployment regions, like the possibility of disabling some of them or to set some budget alert over these regions, however, both this solution can present some limitation: in the first case, the disabling will be total for the entire account, so it could not be possible for any project to use it, while with the policy control it will be possible to modify the available regions list based on the project; the settable alert depends on the budget of the account and when they are generated, many costs are already been performed.

Via the policy pack, it is possible to limits the deployment regions to a reduced set of authorized ones, so that all the resources that will be instantiated must be within them. Thanks to this policy, it will be possible to reduce erroneous deployments.

To verify how would change cloud costs with the deployment across different regions, a simulation of the costs will be executed in some example scenarios.

Since the cloud costs can vary based on the typology of the resources and their configuration, for this simulation three different resources with a specific configuration have been selected so that they will be the same for each region:

- A virtual machine with:
    - 2 vCPU
    - 8 GB of RAM
- A storage account with:
    - 1 TB of memory
- A lambda function with:
    - 1 GB of allocated RAM
    - 10.000.000 requests per month
    - 100 ms of execution per request

All the on-premises version will be evaluated with 730 Work/Hr for the Virtual Machine and 1000 GB of memory for the storage.

Moreover, all the resources will be simulated with 5 configurations: an on-premises, a 1-year saving plan and a 3-year saving plan within the same Europe West region, and the US West and Australia Central regions.

Microsoft Azure resources	Europe West with On-premises resources	Europe West with 1-year saving plan	Europe West with 3-year saving plan	US West with On-premises resources	Australia Central with On-premises resources
VM: D2 v3	79,17€/mo	68,57€/mo	54,11€/mo	77,19€/mo	82,47€/mo
Storage: Blob storage	19€/mo	15,64€/mo	12,59€/mo	10,95€/mo	19€/mo
Lambda: AZ functions	10,83€/mo	9,12€/mo	8,05€/mo	10,83€/mo	10,83€/mo

Table 6 – Azure regions prices comparison with 3 fixed resources

Amazon Web Services resources	Europe West with On-premises resources	Europe West with 1-year saving plan	Europe West with 3-year saving plan	US West with On-premises resources	Australia Central with On-premises resources
VM: t2.large	69,90€/mo	56,59€/mo	40,50€/mo	76,56€/mo	81€/mo
Storage: S3 standard	21,85€/mo	18,14€/mo	15,95€/mo	24,7€/mo	23,75€/mo
Lambda: lambda	13,01€/mo	11,12€/mo	9,08€/mo	13,01€/mo	13,01€/mo

Table 7 - AWS regions prices comparison with 3 fixed resources

As it is possible to notice, all prices depend on both the typology of the resource and the region selected: Lambda function do not change their prices across various regions, while virtual machines and storage account can perform high differences. For example, if the

Europe West region was designed as default region thanks to the application of a saving plan, the execution of VMs in any other region will always perform additional costs.

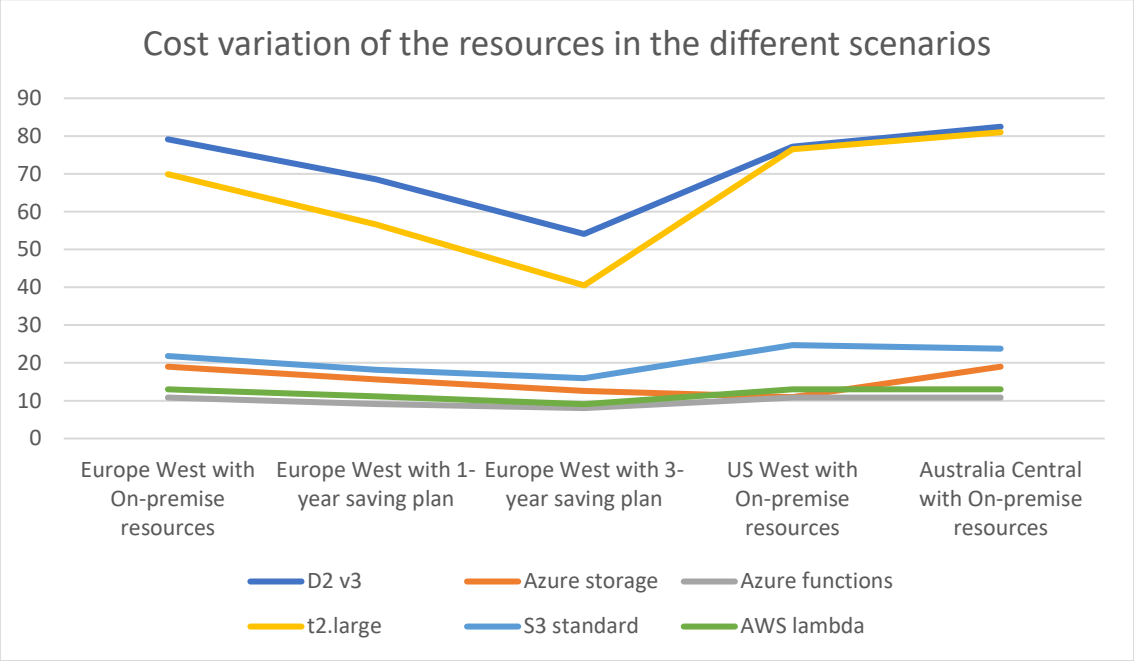


Figure 19 - Cost variation of different resources over multiple scenarios

However, as can be seen from this data, the implementation of this policy does not always lead to cost reductions, since a more economically convenient region could be excluded from the available one. On the other hand, technical choices based for example on the network latency for certain regions may influence the implementation of this policy.

Consequently, the selection of the available regions via both technical and economic factors, represent an important prerogative to the application of this policy.

5.3 Budget and forecasted anomalies detection validation

The third main process developed in this thesis work is the Multi Cloud budget management and anomalies detection via forecasted spent alert.

This process is composed by administrative management operation, like Multi-Cloud budget creation, modification and cancellation, by policy guardrail to avoid the exceeding of the budget limit, and by Cloud Providers alert activation for forecasted spent exceeding. Moreover, all these features have been developed for both general and per project budgets.

For the validation of these features two different operations have been conducted: in the first one the validation of the management operations has been conducted with the alert settings and the verification of their functioning; in the second one the validation of the policies has been checked to verify the guardrail execution.

All the operations have been performed over the Liquid Reply's AWS and Azure subscriptions.

In the first operation a global budget named "*Test_budget*" have been created with a value of 100 $ divided equally over the two cloud providers. Then the budget has been created by the Pulumi Project on both AWS and Azure Cloud Providers. Following, an update of that budget to a value of 10 $ has been performed, verifying all these operation via the management consoles of the Cloud Providers. Since the budget was set on the company's accounts, waiting a period of time without personally deploying any resource, the alerts have been generated to my personal mail address guaranteeing the functioning of them. To end the test, the cancellation of the budget has been performed.

When the alerts have been generated, the forecasted spent of the test budget had been exceeded by projects that was already deployed. However, using a project budget, the identification of the resources that caused the exceeding of the limit would have been easier, and corrective action could be more immediate.

As reported in the 2023 by the FinOps foundation [18], anomalies detection still represents a high difficulty for companies, since only the 17% of them was able to detect anomalies in hours, while the 50,5% took days. Thanks to this automated process, all anomalies that unexpectedly increase the forecasted spending can be easily tracked and detected within hours from their deploy.

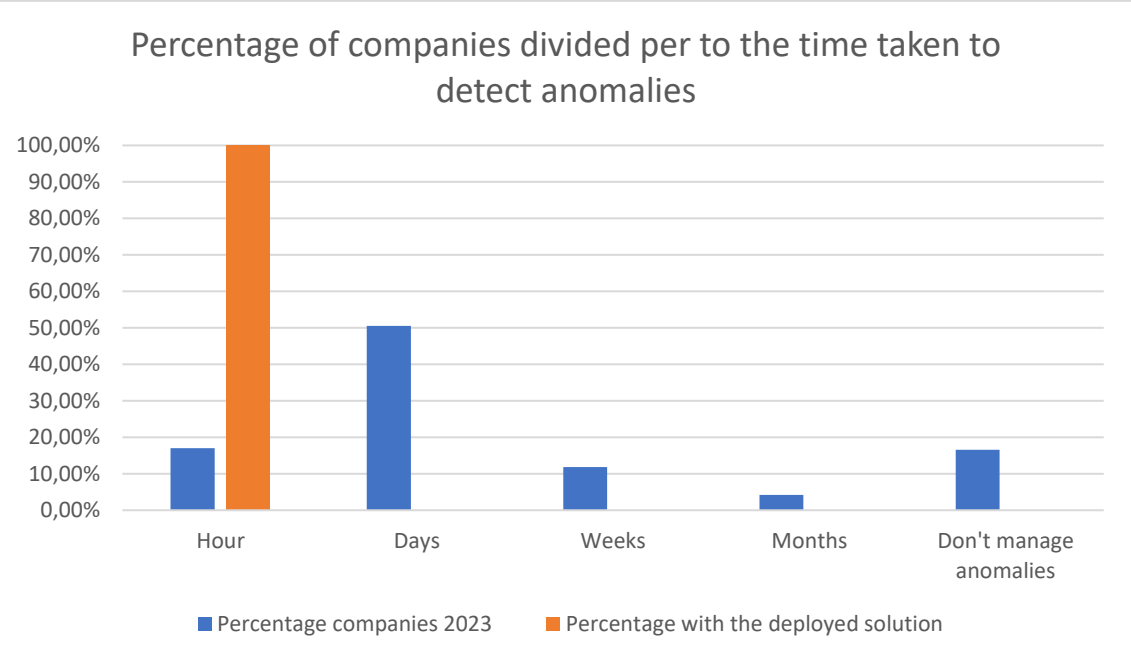


Figure 20 - Percentage of companies divided per time taken to detect anomalies

In the second operation, a complete budget connected to the company account have been created for two weeks. During this period, the deploy of a simple project has been performed many times until the policy prevent it. On the third attempt the forecasted spent limit had been exceeded and even if the deploy was still performable a warning was generated after it, while on the seventh attempt the budget limit was exceeded, and the policy prevented the deployment of the resources.

5.3.1 Anomalies detection via automatic report generation

A further consideration on the anomalies' detection can be carried out, analysing the automated report generation. This process aims to detect all the anomalous resources that are either untagged or deployed in different regions with respect to the default one, reporting also their cost in the last 14 days.

With this report, which can be generated periodically based on the decision of the administrator, also other anomalies can be detected.

The validation of the automated report generation, has been performed over the complete infrastructure of the company, generating one report per day for one week. During this period 10 resources without tag and 10 resources in external regions have been deployed to verify if the automated process reported them. Since this process used the native APIs of the Cloud Providers and does not depend over Pulumi resources management, all of

the forced anomalous resources had been detected thanks to the direct access to the AWS Cost Resources tool and the Azure resource groups tool.

5.4 Performance metrics of the deployment pipeline

In this section, a performance evaluation of the automation deployment pipeline timing over each stage will be conducted.

For the execution of the experiment, the test environment was structured in: a Windows computer with 4 CPU cores and 16 GB of RAM, hosting a Linux Virtual Machine with 4 vCPUs and 8 GB of RAM, on which was installed a Jenkins server. In the Jenkins server, both Pulumi and Python was installed to perform the execution of the Pulumi scripts. As network connection, the computer was connected via Wi-Fi to a router with a 1 Gbit connection. All the experiment results can depend on the performance of the hardware, the network used, and the network latency between the physical location, in Turin, and the selected regions of the Cloud Providers.

For this experiment, four different projects have been deployed via the pipeline, analysing their execution differences:

- A project deployed over a single cloud provider, AWS, composed by a single Standard storage S3 instance.
- A project deployed over a single cloud provider, Azure, composed by a Resource Group and an Azure Storage account.
- A multi cloud providers project composed by an AWS S3 storage, an Azure Resource Group and an Azure Storage account.
- A Multi-Cloud providers project composed by two web servers one deployed in AWS and the other in Azure, principally composed by:
    - An AWS EC2 instance
    - An AWS EC2 Security Group
    - An Azure Resource Group
    - An Azure Virtual Network
    - An Azure Compute Virtual Machine

Following the execution of all the pipelines, it was found that all the stages about the verification of the updates, the checkout of the code and of the policies, the stack

configuration, and also the automated tag application had a comparable timing, while the installation stages and the execution one, had different timing among the various execution.

Regarding the installation stages this is due to the differences between the necessary packages, since for example for single cloud project, all the packages of the second cloud provider will not be installed or even for bigger project where external libraries are used, further installation are needed.

Regarding the execution time, the variation in the timing depends on the Pulumi deploy of all the project. As can be evinced by the data shown in Table 8, bigger project needs more time for the deploy, since more operation will be performed by Pulumi both for the validation via Policy Pack and for the deploy of the resources on the Cloud Providers.

	Single-Cloud AWS	Single-cloud Azure	Multi-cloud Storage	Multi-cloud Web Server
Instantiated Resources	2	3	5	21
Execution time	25 sec	47 sec	1 min 54 sec	4 min 43 sec

Table 8 - Execution time of the deploy of the different projects

5.5 Comparison with existing FinOps tools

All the developed processes can be placed among some existing tools, to understand better their scope.

Even if each existing solution has been created for a specific problem, all of them have in common the application of some FinOps practices to help companies in the management of cloud costs.

In this section, a comparison between the processes developed and the various tools will be made, understanding their advantages and limitations.

In the following Table 9, a comparison matrix has been obtained comparing the FinOps practices that each solution deals with.

	Developed processes	Apptio solution	CloudSaver solution	Infracost solution	CloudBolt solution
Tag automation	X		X	X	
Tag Policy	X		X	X	
Budget management	X	X			X
Budget Guardrails Policy	X			X	X
Deployment Region Policy	X				
Anomalies detection	X	X			
Automated Reporting generation	X	X		X	X
Works with IaC	X			X	
Works with Pulumi	X				
Script real time cost evaluation				X	
Handle multiple cloud providers	X	X		X	X
Own management application			X		X
Pay per use		X	X	X	X

Table 9 - Comparison matrix between FinOps existing tools and the developed processes

Analysing the obtained results, it is possible to notice that some tools prefer to focus on a specific FinOps problem, for example the CloudBolt and Aptio solutions focuses over budget and report management, while the CloudSaver solution over Tag management and

application. However, FinOps execution cannot be limited to only a specific problem, and a more configurable solution is needed by companies to manage their specific needs.

For instance, for the application of FinOps practices in a more comprehensive way, over the Infrastructure as Code approach only the Infracost solution exists, and it is limited to a Doman Specific Language for Terraform even if it is able to offer a VsCode extension for real time cost evaluation of the projects.

In this mode, the developed processes represent an innovation for these practices, since each company will be able to configure a more complete and still expandible solution that will help in a proactive and automated execution of FinOps practices.

6 Use case definition

In this chapter, a use case definition will be described for the application of the FinOps developed processes in a real scenario.

After a first description of the stakeholders involved in the usage of the solution and a presentation of all the requirements, the definition of the methodology of application of the processes will be described. The scope of this analysis will be a more precise and complete explanation, which will help in understanding the functioning and the actual application over a complex infrastructure.

Moreover, the developed processes can have multiple usage modalities, since it is possible to use them as single processes or in an integrated form. In this chapter, the integrated version will be explored, since in case of singular uses, their application will depend on the current infrastructure where they will be applicated.

6.1 Use case scenario stakeholders

The complex integrated version of the processes aims at the achievement of different operations since via this solution users will be able to: develop a multi cloud infrastructure; manage multi cloud budgets and Tag account definitions; set policy guardrails to projects; generate automated report about the multi cloud infrastructure.

Since this solution defines a precise operational methodology inside a company, multiple different users will be able to interact with it.

The main stakeholders involved in the process are three: developers, FinOps operators, administrators. All of them will be part of the FinOps team, since one aspect of the FinOps application is the creation of a unified team to increase communication and collaboration; however, each of them will have specific objectives and privileges across the system, based on their role.

The developers, which will be responsible for the creation and modification of the cloud infrastructure, will need to interact with the system for the deploy of their code.

FinOps operators, will be responsible for the management of all the policy, budgets, tags and the analysis of the automatic generated report or anomalies detection to apply corrective actions on time.

The administrator will be able to manage all the users in the system, the accounts on the Cloud Providers, and also interact with FinOps operators for the analysis of the automatic generated report.

6.2 Use case scenario requirements

For the deployment of the processes, some technical requirements are needed. Since different technologies are involved in the processes, each of them will require a specific configuration:

- For the execution of the Pipeline, the Jenkins server need to be instantiated over a remote and accessible machine connected to the network.
- To perform the deploy of the infrastructure, the Pulumi software need to be installed on the same server of Jenkins since the Pulumi engine will be activated by Jenkins for the deploy.
- Three external accounts will be needed by Jenkins, to interact with Pulumi, AWS and Azure. Each account will allow Jenkins to perform the deploy of the infrastructure:
    - *Pulumi credentials*: they will be used to connect to the Pulumi Cloud. Via them also the FinOps operators and administrators will be able to control the deployment state of all the infrastructure.
    - *AWS credentials*: they are composed by the AWS_ACCESS_KEY_ID that represent the user identification and the AWS_SECRET_ACCESS_KEY which is the secret to authenticate the user.
    - *Azure credentials*: they are composed by four components and allows applications to interact with the service. There are both the ARM_CLIENT_ID and the ARM_CLIENT_SECRET that authenticate the user, the ARM_TENANT_ID that identifies the tenant which represent the organization, and the ARM_SUBSCRIPTION_ID that identifies the specific subscription where the user is registered to.
- For the execution of the automated reporting generation, a Python compiler needs to be installed on the same Jenkins server.
- Two external repositories, for example GitLab repositories, have to be configured. One for the administrative part where user tag information, policy definitions and

the automatic reports are stored. The other for the deployment part where developers can store their code.

6.3 Use case scenario methodology

Once all the involved users have been defined with also the necessary configurations, a description of the use case scenario can be performed.

Firstly, the administrator should activate the Jenkins server, install Pulumi and Python, and export in Jenkins all the configuration keys for Pulumi AWS and Azure. After he have to define the accounts for all the users on Jenkins. Since it is possible to define different permissions for the Pipelines, a correct configuration will assign to developers the possibility of executing only the AutoTaggablePipeline, without the permission for its modification, while for the FinOps operators, full permissions will be granted to all the pipelines.

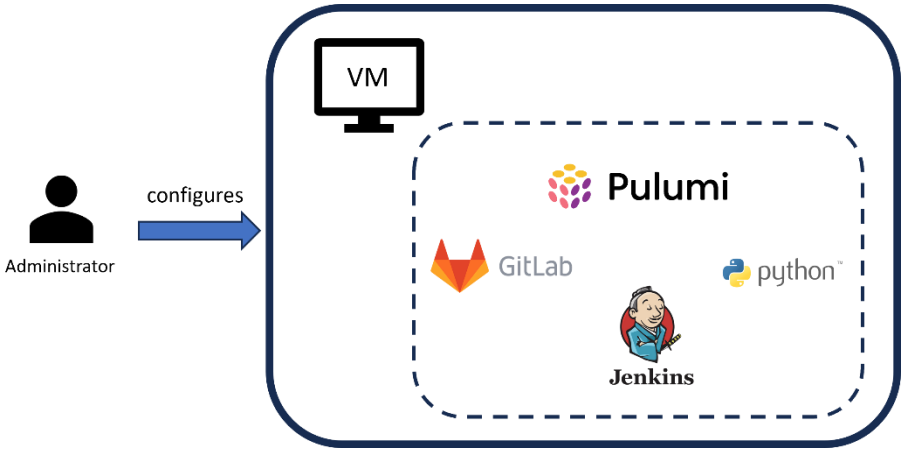


Figure 21 - Administrator configuration operation

Following, the administrator should create and configure the two repositories, giving access to one of them also to developers, while to the other only he and FinOps Operator will be able to access. Then, he should set these repositories as the pipeline source for both project and policies. Finally, he will be able to manage user tag configuration file via a Jenkins Pipeline.

After the configuration phase is completed, developers will be able to deploy their code: since there are no constraints for the selection of the integrated development environment (IDE), they can write the code in any supported language by Pulumi and push it on the

repository. This action will trigger the execution of the autoTaggablePipeline, and the results can be controlled by the developers via their Jenkins account.

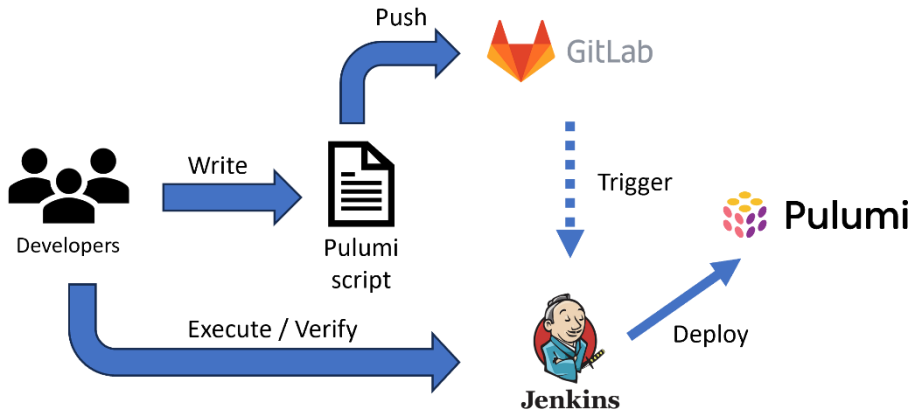


Figure 22 - Developers interaction with the processes

For the management operations, FinOps operators will be able to use the Jenkins Pipelines to do every operation, using the created account to manage budgets and user tags. Also, once everything has been set, they can activate the automated reporting generation, setting also the period of time every which it will be executed.

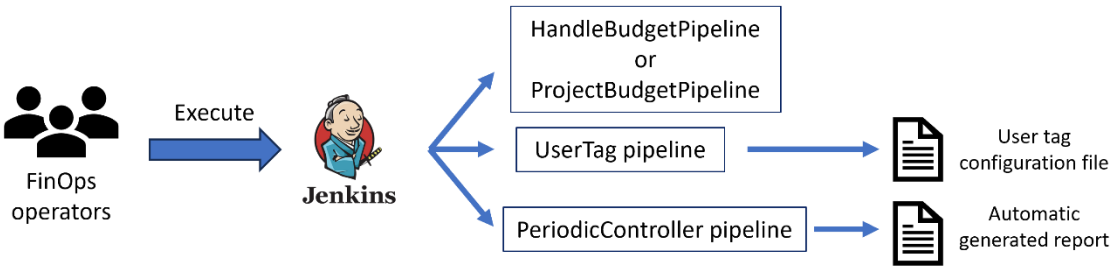


Figure 23 - FinOps operators pipeline execution schema

7 Study limitation and possible future developments

In this chapter, an examination of the limits of the developed solution will be carried out. Therefore, an analysis on possible future improvements will be presented.

The developed solution is composed by different processes that, individually or as a whole will resolves specific FinOps challenges and will help companies in the achievement of the FinOps capabilities. However, two different aspects can be considered: the possibility of extending the solution to cover others FinOps practices and alternative approaches for a different handling of already covered ones.

FinOps practices span across different aspects from financial to code management and the application of all of them, will requires many time and resources. For this reason, in this thesis work a reduced set of them have been selected following an analysis of what in the actual state was more required. Moreover, all the solutions are thought to be configurable to each company's need, since FinOps practices application depends on each specific necessity.

A first limitation of the processes can be represented by the available Cloud Providers where they can be used: since their aim was to understand the feasibility of the application of FinOps practices, the thesis work focussed only on two cloud providers, and by utilization rate have been selected Amazon Web Services and Microsoft Azure. However, a future study can extend the solution to other multiple cloud providers to analyse the interaction within the solution in a group with more than 2 cloud providers.

A second limitation is represented by the selected policy deployed. Since FinOps practices are something that depends on the company's need, a reduced set was deployed focussing on general aspects that resolves some challenges. However, it can be expanded with other policies that can perform other controls over a multi cloud infrastructure: for example, a future policy can be able to execute a cost prevision based on actual cost of the resources, to advertise proactively if the infrastructure will exceed the defined budget. Additionally, some policies could have been manged differently: the forecasted spent policy could have been set to interrupt the deploy of the infrastructure instead of only

generate an alert; the region policy could have been set differently, for example generating a warning if a different region had better prices.

A third limitation concern about the report generation since its focus was on untagged resources and resources deployed in forbidden regions. In a future extension other information about the Multi-Cloud infrastructure could be provided by the report: for example, it will be possible to extend this report with also information regarding all the budgets of the various cloud providers with statistical computations. Moreover, since the report depends on the tag process, in an alternative version of this process where a different tag management could have been structured, there is the necessity of a modification of the automated reporting generation, creating alternative solution for anomalies detection.

Regarding new processes that can extend the coverage of the solution over FinOps practices, many aspects can be taken into account. Furthermore, the two most interesting developments will be exposed: an automated cost/benefit evaluator that can also perform infrastructure resizing; a cost prevision process that can provide more information for the budget definition.

The first development can be created as an automated pipeline that runs periodically, and that for each project perform the evaluation. If costs overcome the resources necessity, a shrinkage of the infrastructure will be performed, whereas if the resources necessity is higher than the defined costs, an enlargement of the infrastructure will be performed, all in an automated manner.

The second deployment can be created as an extension for Integrated Development Environment (IDE) or as a pipeline that can perform an evaluation of the entire project, retrieving the actual cost of all the resources and providing some evaluation about the cost of the deploy per month. Moreover, it can also provide information about the economic convenience about the deploy of a resource on one Cloud Provider with respect to the others.

However, consequently to these extensions, a presentation of the advantages that are performed will be needed to understand if their application satisfies the FinOps challenges and the achievement of the FinOps capabilities.

8 Conclusions

In this thesis work, different processes have been analysed, developed and tested to resolve some FinOps challenges, applicate them to an Infrastructure as Code methodology, and create a base solution for the FinOps practices application in a company's structure.

All these processes represent a part of a more complex solution, which aims to bring proactivity and automation into company's practices, allowing a complete automated and controlled tag management and application, an integrated multi cloud total and per project budgets management, and an automated report generation that will help in the anomalies' detections.

To achieve all the results, different technologies have been involved. Since the main process focuses on the Infrastructure as Code approach, the newly Pulumi technology has been selected thanks to its innovation in coding the infrastructure via generic languages and not a domain specific language (DSL). Indeed, all the scripts developed has been coded in Python.

The other technologies involved, like Jenkins and GitLab, support all the processes via the implementation of a continuous control or automated pipeline and via the usage of structured repositories.

In conclusion, in the present days the FinOps approach still requires a lot of investments by companies to reach ah higher grade of maturity, and all these processes will help companies in improving their FinOps application. Moreover, since Cloud Computing is a continuously evolving practice where its direction is based on Multi-Cloud environments that requires increased management and companies are even more focusing on their cloud costs, FinOps practices will evolve specularly to the Coud Computing, bringing new innovations. This thesis work will provide a concrete support for the creation of these new processes demonstrating the feasibility and the advantages of their application.

Bibliography

[1] «Informazioni su cloud pubblici, privati e ibridi,» [Online]. Available: https://azure.microsoft.com/it-it/resources/cloud-computing-dictionary/what-are-private-public-hybrid-clouds. [Consultato il giorno 05 09 2023].

[2] «IaaS vs. PaaS vs. SaaS,» RedHat, 16 08 2022. [Online]. Available: https://www.redhat.com/en/topics/cloud-computing/iaas-vs-paas-vs-saas. [Consultato il giorno 08 09 2023].

[3] «Aree geografiche e zone,» Google, [Online]. Available: https://cloud.google.com/compute/docs/regions-zones?hl=it. [Consultato il giorno 10 09 2023].

[4] «Regions and Zones,» [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html. [Consultato il giorno 10 09 2023].

[5] S. J. Bigelow, «Select the right cloud regions, availability zones to optimize costs,» TechTarget, 24 03 2017. [Online]. Available: https://www.techtarget.com/searchcloudcomputing/tip/Learn-the-cost-implications-of-cloud-regions-and-availability-zones. [Consultato il giorno 15 09 2023].

[6] F. Fregi, «Google Cloud apre la seconda region in Italia. Da oggi disponibile l'infrastruttura di Torino,» 23 03 2023. [Online]. Available: https://blog.google/intl/it-it/prodotti/cloud/google-cloud-apre-la-seconda-region-in-italia-da-oggi-disponibile-linfrastruttura-di-torino/. [Consultato il giorno 11 09 2023].

[7] «Microsoft announces its first cloud region in Italy, accelerating innovation and economic opportunity,» 05 06 2023. [Online]. Available: https://news.microsoft.com/europe/2023/06/05/microsoft-announces-its-first-cloud-region-in-italy-accelerating-innovation-and-economic-opportunity/. [Consultato il giorno 11 09 2023].

[8] F. Richter, «Amazon Maintains Lead in the Cloud Market,» Statista, 08 08 2023. [Online]. Available: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/. [Consultato il giorno 15 09 2023].

[9] L. Goasduff, «Gartner Top 10 Trends in Data and Analytics for 2020,» Gartner, 19 10 2020. [Online]. Available: https://www.gartner.com/smarterwithgartner/gartner-top-10-trends-in-data-and-analytics-for-2020. [Consultato il giorno 10 09 2023].

[10] T. Luxner, «Cloud computing trends and statistics: Flexera 2023 State of the Cloud Report,» Flexera, 05 04 2023. [Online]. Available: https://www.flexera.com/blog/cloud/cloud-computing-trends-flexera-2023-state-of-the-cloud-report/. [Consultato il giorno 10 09 2023].

[11] «What is Infrastructure as Code (IaC)?,» 11 05 2022. [Online]. Available: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac. [Consultato il giorno 10 09 2023].

[12] F. Pialoux, «Best Infrastructure as Code Tools (IaC): The Top 11 for 2023,» [Online]. Available: https://bluelight.co/blog/best-infrastructure-as-code-tools. [Consultato il giorno 20 09 2023].

[13] «Pulumi vs Terraform,» [Online]. Available: https://medium.com/datamindedbe/pulumi-vs-terraform-choosing-your-iac-tool-6d17b5222545.

[14] «How Pulumi works,» [Online]. Available: https://www.pulumi.com/docs/concepts/how-pulumi-works/. [Consultato il giorno 16 09 2023].

[15] "Pulumi Cloud Registry," [Online]. Available: https://www.pulumi.com/registry/. [Accessed 11 September 2023].

[16] «FinOps principles,» [Online]. Available: https://www.finops.org/framework/principles/. [Consultato il giorno 15 09 2023].

[17] «The REAL State of FinOps,» [Online]. Available: https://resources.cloudbolt.io/industry-reports/the-real-state-of-finops.

[18] «The State of FinOps,» [Online]. Available: https://data.finops.org/. [Consultato il giorno 15 09 2023].

[19] «Resource Utilization & Efficiency,» [Online]. Available: https://www.finops.org/framework/capabilities/utilization-efficiency/. [Consultato il giorno 15 09 2023].

[20] A. Pathak, «Pros and Cons of Multi-Cloud vs. Single-Cloud Environments,» 05 2022. [Online]. Available: https://medium.com/illuminations-mirror/pros-and-

cons-of-multi-cloud-vs-single-cloud-environments-ffd0f52a10ec#:~:text=Multi%2Dcloud%20environments%20offer%20flexibility,vendor%20lock%2Din%20and%20dependency.. [Consultato il giorno 16 09 2023].

[21] «What is FinOps?,» 11 2021. [Online]. Available: https://www.finops.org/introduction/what-is-finops/. [Consultato il giorno 17 09 2023].

[22] A. Behera, «The Role of Automation in FinOps: Best Practices for Cloud Cost Management,» CloudKeeper, 05 05 2023. [Online]. Available: https://www.cloudkeeper.ai/insights/blog/role-of-automation-in-finops. [Consultato il giorno 17 09 2023].

[23] «Workload Management & Automation,» [Online]. Available: https://www.finops.org/framework/capabilities/workload-management-automation/. [Consultato il giorno 14 09 2023].

[24] «Pulumi policy as code concepts,» [Online]. Available: https://www.pulumi.com/docs/using-pulumi/crossguard/core-concepts/. [Consultato il giorno 17 09 2023].

[25] «Cloud Financial Management,» [Online]. Available: https://www.apptio.com/solutions/cfm/. [Consultato il giorno 19 09 2023].

[26] «CloudSaver,» [Online]. Available: https://www.cloudsaver.com/. [Consultato il giorno 15 09 2023].

[27] «Infracost,» [Online]. Available: https://www.finops.org/members/infracost/. [Consultato il giorno 15 09 2023].

[28] «CloudBolt,» [Online]. Available: https://www.cloudbolt.io/cloud-cost-management/. [Consultato il giorno 15 09 2023].

[29] «Boto3 documentation,» [Online]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/ce/client/get_cost_and_usage_with_resources.html. [Consultato il giorno 19 09 2023].

[30] U. A. Agarwal, «Azure Subscriptions,» 30 06 2023. [Online]. Available: https://k21academy.com/microsoft-azure/az-900/az-900-azure-subscriptions/#:~:text=Azure%20Subscriptions%20are%20a%20logical,resources%20used%20in%20that%20account.. [Consultato il giorno 19 09 2023].

[31] «Resource option: transformations,» [Online]. Available: https://www.pulumi.com/docs/concepts/options/transformations/. [Consultato il giorno 19 09 2023].

[32] «Pipeline Syntax,» [Online]. Available: https://www.jenkins.io/doc/book/pipeline/syntax/.

[33] «The State of Cloud Tag Management 2022,» [Online]. Available: https://www.cloudsaver.com/resources/white-papers/state-of-cloud-tag-management-2022/. [Consultato il giorno 10 08 2023].

[34] «Resource types you can use with AWS Resource Groups and Tag Editor,» [Online]. Available: https://docs.aws.amazon.com/ARG/latest/userguide/supported-resources.html#supported-resources-tagging-console. [Consultato il giorno 18 07 2023].

[35] «Supporto dei tag per le risorse di Azure,» 25 03 2023. [Online]. Available: https://learn.microsoft.com/it-it/azure/azure-resource-manager/management/tag-support. [Consultato il giorno 18 06 2023].