**10**

**Managing Costs with FinOps**

Building and running a data center in public clouds can be more cost-effective than going the traditional way. However, running a data center in public clouds still costs money. Hence, as ever, you still need ways to control costs. **financial operations**(**FinOps**)—is all about cost control.

This chapter focuses on the starting point for managing FinOps in multi-cloud environments: the provisioning of resources and the costs that come with the deployment of resources. We will learn how to keep track of these costs in the public clouds of Azure, AWS, GCP, Alibaba Cloud, and **Oracle Cloud Infrastructure** (**OCI**). However, we start with a brief introduction to the principles of FinOps.

In this chapter, we’re going to cover the following topics:

* Understanding the principles of FinOps
* Defining guidelines for provisioning cloud resources
* Defining cost policies for provisioning
* Understanding account hierarchy
* Understanding license agreements
* Defining tagging standards
* Validating and managing billing

**Understanding the principles of FinOps**

For starters, FinOps is not about saving money in the cloud, but about making money with the cloud. Hence, it should be tightly integrated into business planning and your architecture. FinOps has proven to be a must, with the advent of the cloud and DevOps, for staying in financial control.

Why would an enterprise need FinOps for its transition to the cloud? The simple answer to that question is: you need to have visibility into cloud spending. Achieving that sounds a lot easier than it actually is. In the cloud, engineers have the power to deploy resources “at will.” There’s no need to order new equipment, since the “equipment” is there and ready to use. While a finance department would need to sign off investments for hardware, they hardly know what’s going on in cloud environments. Above all, with DevOps and multiple releases per month, week, or even day, cloud consumption is very dynamic. That makes it hard to keep track of spending. Moreover, it makes it hard to define whether the spending is contributing to the business goals—which it should. Enterprises need a methodology to track investments and enable them to relate investments to business goals, validating the business case. The FinOps Foundation, part of the Linux Foundation, has designed such a methodology. What FinOps tries to achieve is this:

* Inform
* Identify optimization targets
* Operationalize changes to realize the optimization targets

This is an iterative process, meaning that once a company has gained insights as to their cloud spending, they can identify possible targets where consumption can be optimized. The next step is to think about solutions to optimize the usage of cloud resources. The intent is not to cut costs per se, but to get more benefit out of cloud usage. In FinOps, this is an iterative approach, meaning it is not assumed that a company will do everything in one go. Teams start with a **minimal viable product** (**MVP**), build capabilities as they learn more, and then improve. The FinOps Foundation calls this lifecycle approach “**crawl, walk, run**.” Teams will improve as they go through this lifecycle.

The most important principles of FinOps can be summarized as follows:

* Define principles and guidelines for the consumption of resources
* Define principles and guidelines for the provisioning of resources
* Define asset tagging standards
* Validate and manage billing
* Use chargeback models
* Ensure efficient purchasing options
* Carry out license management
* Automate

In the next sections, we will discuss these principles in more detail and explore examples of how they work with the major cloud providers.

**Define guidelines for the provisioning of cloud resources**

Before we dive into cost control in the provisioning of resources, we need to understand how resource provisioning works in the public cloud. There are lots of different ways to do this, but for this chapter, we will stick with the *native* provisioning tools that cloud providers offer.

There are basically two types of provisioning:

* Self-provisioning
* Dynamic

Typically, we start with self-provisioning through the portal or web interface of a cloud provider. The customer chooses the resources that are needed in the portal. After confirmation that these resources may be deployed in the cloud environment, the resources are spun up and made available for usage by the provider.

The resources are billed by hour or minute unless there is a contract for **reserved instances**. Reserved instances are contracted for a longer period—1, 3, or 5 years. The customer is guaranteed availability, capacity, and usage of the pre-purchased reserved instances. A benefit of reserved instances can be that cloud providers offer discounts on these resources. Over a longer period, this may be very cost-effective. It’s a good way to set budget control: a company will know exactly what the costs will be for that period. However, it’s less flexible than a pay-as-you-go model and, even more important, it requires up-front payments or investments, though most providers also offer the option to choose monthly payments at no extra cost.

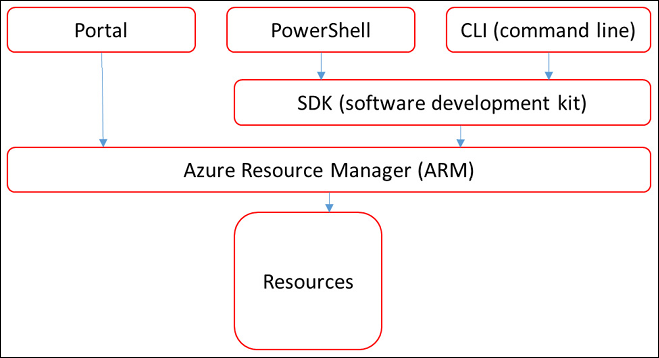
Dynamic provisioning is more of an automated process. An example is a web server that experiences a spike in load. When we allow automatic scale-out or scale-up of this web server, the cloud provider will automatically deploy more resources to that web server or pool. These extra resources will also be billed at a pay-as-you-go rate.

Let’s look at how these resources are deployed.

**Deploying resources in Azure using ARM**

Azure works with **Azure Resource Manager** (**ARM**), Azure’s solution for **Infrastructure as Code** (**IaC**). ARM is a service that handles requests from users and makes sure that the requests are fulfilled. That request is sent to ARM. Next, ARM executes all the actions to actually deploy a VM, for example. What it does is assign memory, the processor, and disks to a VM and make it available to the user. ARM can do this with all types of resources in Azure: VMs, storage accounts, web apps, databases, virtual network resource groups, subscriptions, management groups, and tags.

ARM can be directly accessed from the portal. However, most developers will be working with PowerShell, the **Azure Command-Line Interface** (**CLI**), or REST APIs. In that case, the request goes to a **software development kit** (**SDK**). SDKs are libraries with scripts and code templates that can be called through a command in PowerShell or the CLI. From the SDK, the resources are deployed in ARM. The following diagram shows the high-level conceptualization for ARM:



*Figure 10.1: High-level concepts for ARM*

In the next section, we will learn how to deploy resources in AWS.

**Deploying resources in AWS using CloudFormation and OpsWorks**

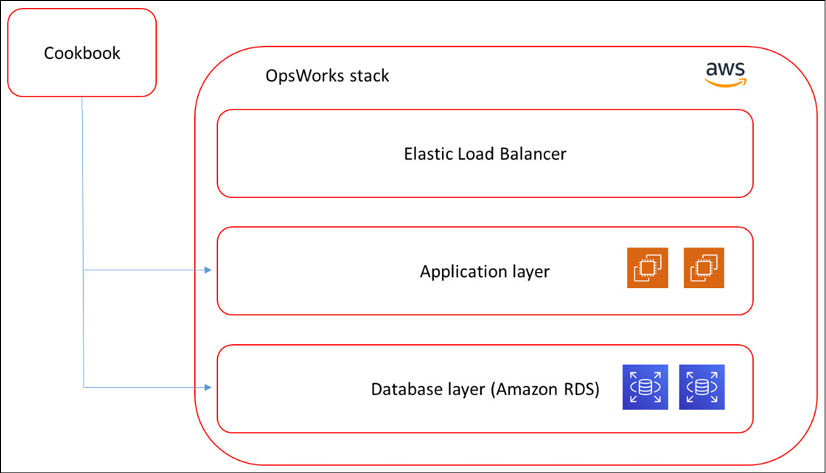
With AWS, we have to discuss two possible solutions to provision workloads. The first one is CloudFormation. As with ARM in Azure, CloudFormation is IaC, using JSON or YAML templates in a central repository, typically an S3 bucket in AWS. The code can be accessed through the browser console or CLI, allowing the architect or engineer to provision infrastructure from these templates in the designated AWS environments.

Another solution is AWS OpsWorks for the automatic deployment and configuration of resources. It works with a cookbook repository—a term that AWS borrowed from the automation tool Chef. That makes sense since, under the hood, OpsWorks works with Chef and creates managed instances of Chef and Puppet.

AWS OpsWorks Stacks allows for the automatic deployment of an entire stack, including infrastructure, load balancing, database, and the application itself. The stack itself is the core component for any deployment in AWS; it’s the construct that holds the different resources, such as EC2, VMs, and Amazon **Relational Database Service**(**RDS**) database instances. OpsWorks makes sure that these resources are grouped together in a logical way and deployed as that logical group—we call this the cookbook or recipe.

It’s important to remember that Stacks works in layers. The first layer is the **Elastic Load Balancing** (**ELB**) layer, which holds the load balancer. The next layer hosts the VMs, the actual servers that are deployed from EC2. The third layer is the database layer. If the stack is deployed, you can add the application from a different repository. OpsWorks can do this automatically as soon as the servers and databases are deployed, or it can be done manually.

*Figure 10.2* shows the conceptualization of an OpsWorks stack:



*Figure 10.2: High-level overview of an OpsWorks stack in AWS*

Next, we will look at Google’s Deployment Manager.

**Deploying resources in GCP using Deployment Manager**

In GCP, the native programmatic deployment mechanism is Deployment Manager. We can create resources and group them logically together in a deployment. For instance, we can create VMs and a database and have them as one code file in a deployment. However, it does take some programming skills to work with Deployment Manager. To start, you will need to have the gcloud command-line tool installed. Next, create or select a GCP project. Lastly, resources are defined in a deployment coded in **Yet Another Markup Language** (**YAML**), which is commonly used. When the deployment is ready, we can actually deploy it to our project in GCP using gcloud deployment –manager.

As mentioned, it does take some programming skills. Deployment Manager works with YAML files in which we specify the resource:

* **Machine type**: A set of predefined VM resources from the GCP gallery
* **Image family**: The operating system
* **Zone**: The zone in GCP where the resource will be deployed
* **Root persistent disk**: Specifies the boot sequence of the resource
* **IP address:**The internet address the resource can be reached at

This information is stored in a vm.yaml file, which is deployed by Deployment Manager.

The final two clouds that we will discuss are Alibaba and Oracle’s OCI.

**Deploying to Alibaba using Terraform**

**Terraform** is a tool to deploy IaC and can be used on every platform that we have discussed so far. Alibaba recommends using Terraform to deploy resources to their platform. Before we do that, we need to have Terraform on our machine. Next, we can define what resources we need in Alibaba, which starts with deploying the **Virtual Private Cloud** (**VPC**) and then the compute engine from **Elastic Cloud Service**(**ECS**)—indeed, the same terminology that AWS uses. We also need to configure networking, using a gateway, and assign IP addresses.

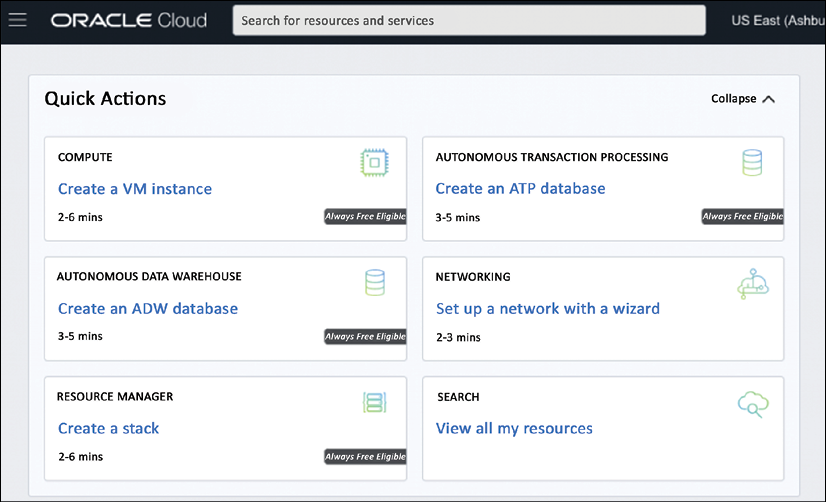
The following step is to check the configurations in the Terraform scripts. Alibaba has demo configurations in Git that contain best practices. The files can be downloaded using git clone: <https://code.aliyun.com/best-practice/017-int.git>. Take note of the fact that the original name of Alibaba Cloud is Aliyun.

From the demo files, you can check the configurations before the resources are deployed to Alibaba, taking the subsequent Terraform steps init, plan, and apply. In the ECS console, we can check if the resources have indeed been deployed.

A good explanation of how to work with Terraform on Alibaba Cloud is provided at <https://www.alibabacloud.com/blog/deploy-alibaba-cloud-resources-by-using-terraform_596412>.

**Deploying resources to Oracle Cloud**

Like Alibaba, OCI Resource Manager also uses Terraform as a deployment tool. Resource Manager can be launched from the main console, where it appears as one of the “quick actions,” as shown in *Figure 10.3*.



*Figure 10.3: Quick access to Resource Manager in Oracle Cloud*

Engineers can use various Terraform sources inside Resource Manager, such as GitHub, Git, and Bitbucket. Next, the engineer or developer creates a stack. For this stack, we can use pre-built Terraform configuration templates for Oracle. The Terraform configuration will automatically prepopulate variables to create the stack:

* Tenancy **Oracle Cloud Identifier**(**OCID**)
* Compartment OCID
* Region
* The OCID of the current user

These values are used when the Terraform commands plan*,*apply*,*anddestroy are triggered. Remember that the Terraform code for the stacks is loaded in OCI Resource Manager.

**Define cost policies for provisioning**

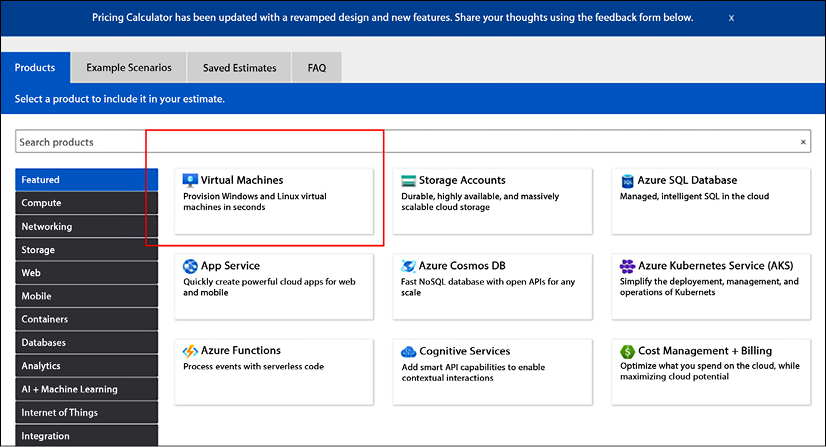
In the previous section, we learned how to provision resources to clouds. This chapter is about keeping control of costs while provisioning resources. In *Chapter 11*, *Maturing FinOps*, we will learn about controlling methodologies and services, but we have to start with estimating our costs before we can set controls.

Let’s start by saying that the sky is the limit in these clouds, but unfortunately, most companies do have limits to their budgets. So, we will need to set principles and guidelines for what divisions or developers are allowed to consume in cloud environments, to avoid budgets being overrun.

**Using the Azure pricing calculator**

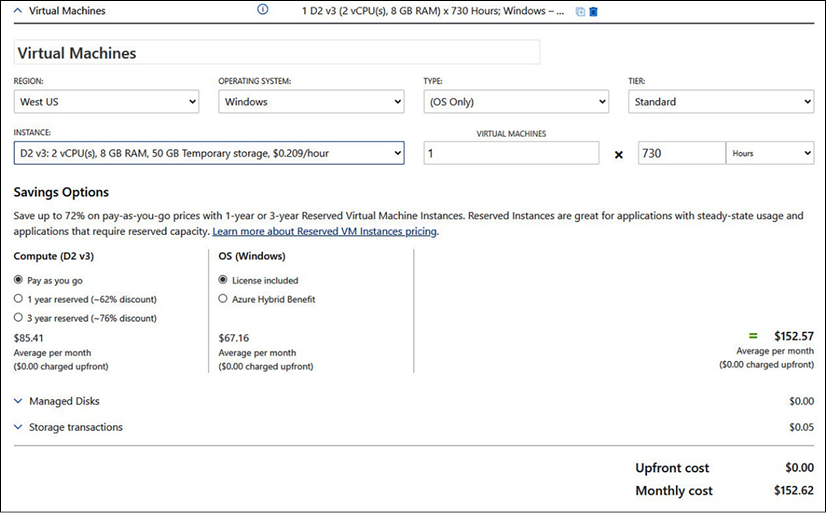
It’s easy to get an overview of what a resource, for example a VM, would cost us in Azure: the pricing overview on <https://azure.microsoft.com/en-us/pricing/calculator/> is a very handy tool for this and is, like all the other calculators and estimation tools that we will explore, completely free of charge to use.

If we open the page, we can see the **Virtual Machines** button, as shown in *Figure 10.4*:



*Figure 10.4: The Virtual Machines option in the Azure pricing calculator*

The portal will display all the possible choices that are offered in terms of VMs, as shown in *Figure 10.5*:



*Figure 10.5: Tab details for VMs in the Azure pricing calculator*

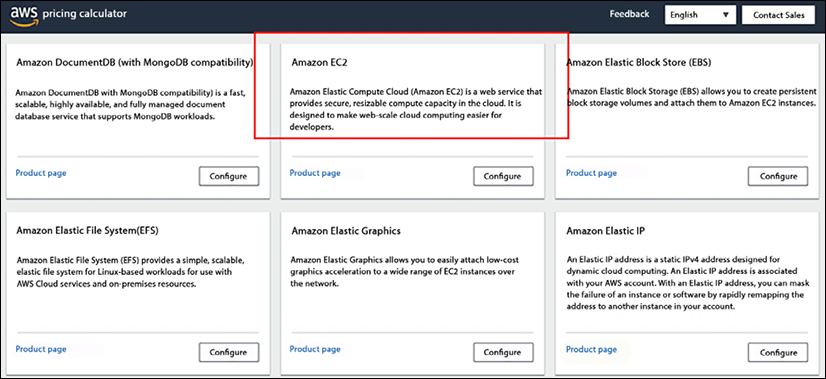
In the screenshot, the **D2 v3** VM has been selected. This is a standard VM with two virtual CPUs and 8 GB of memory. It also comes with 50 GB of ephemeral storage: this is temporary storage that exists as long as the VM runs. We can also see that it will be deployed in the **West US** Azure region, running **Windows** as the operating system. We purchase it for 1 month, or **730** hours, but under the condition of **Pay as you go**—so we will only be charged for the time that this VM is up and running. For the full month, this VM will cost us US$152.62. Note that prices will vary over time.

We could also buy the machine as a reserved instance, for 1 or 3 years. In that case, the VM cost would be reduced by 62% for 1 year and 76% with a 3-year commitment. The reason Azure does this is that reserved instances mean guaranteed revenue for a longer period.

The D2 v3 is a general-purpose machine, but the drop-down list contains well over 130 different types of VMs, grouped into various series. The D-series are for common use. The drop-down list starts with the A-series, which are basic VMs mainly meant for development and testing. To run a heavy workload such as an SAP HANA in-memory database, an E-series VM would be more appropriate. The E64s v4 has 64 vCPUs and 500 GB of memory, which would cost around US$5,000/month. It makes sense to have this type of VM as a reserved instance.

**Using the AWS calculator**

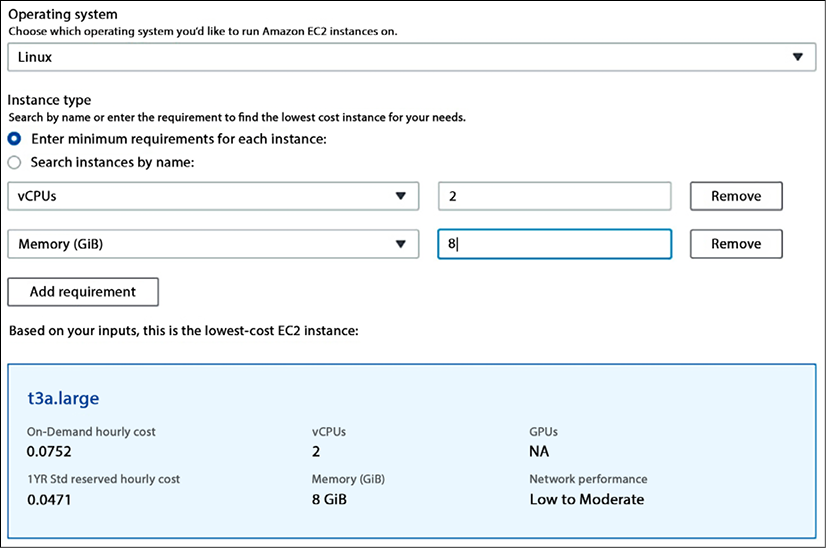
The same exercise can be done in AWS, using the calculator at <https://calculator.aws/#/>. By clicking **Create an estimate**, the following page is shown. Next, select **Amazon EC2** as the service for creating VMs:



*Figure 10.6: Option for EC2 VMs in the AWS pricing calculator*

By clicking on that tab, a similar screen is displayed as in Azure. There is, however, one major difference. In Azure, the VM machine type is also taken into account.

It can be done in AWS, too; the requirements of the machine can be specified by indicating how many vCPUs and how much memory a machine should have. AWS will next decide what type of VM fits the requirements. The following example shows the requirements for a machine with two vCPUs and 8 GB of memory. AWS has defined t3a.large—**t3a** being a specific instance size—as a suitable machine:



*Figure 10.7: Defining specifications for a VM in the AWS pricing calculator*

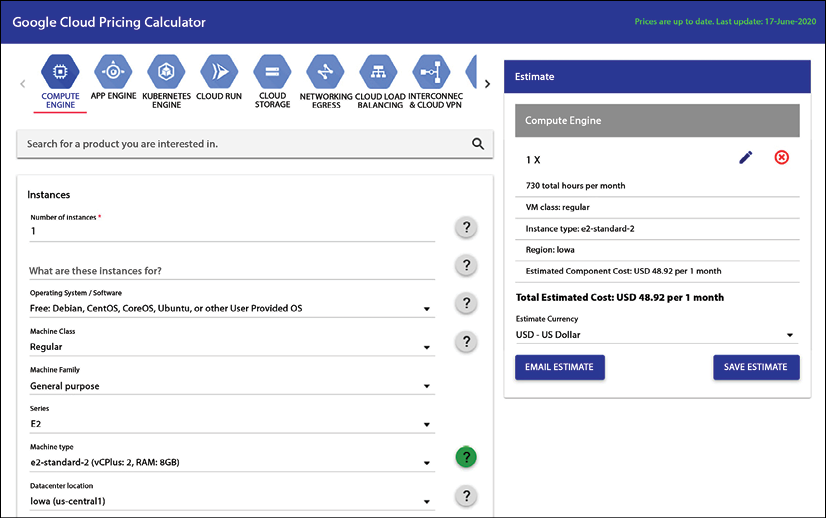
The next decision to make is regarding the cloud strategy. AWS offers on-demand and reserved instances for 1 and 3 years, with the possibility of no, partial, or full up-front payments. With a relatively small VM such as our example, payment wouldn’t become an issue, but also, AWS offers some huge instances of up to 64 vCPUs and up to 1 TB of memory, which would cost some serious money.

That’s the reason for having guidelines and principles for provisioning. We don’t want a developer to be able to *deploy*a very heavy machine with high costs without knowing it or having validated a business case for using this machine, especially since the VM is only one of the components: storage and networking also need to be taken into consideration. Costs could easily rise to high levels.

It starts with the business and the use case. What will be the purpose of the environment? The purpose is defined by the business case. For example: if the business needs a tool to study maps in a geographic information system, then software that views and works with maps would be needed. To host the maps and enable processing, the use case will define the need for machines with strong graphical power. Systems with **Graphics Processing Units** (**GPUs**) will fit best. In Azure, that would be the N-series; these machines have GPUs and are designed for that task. The equivalent in AWS is the G- and P-series, and in GCP and Azure, we can add NVIDIA Tesla GPUs to Compute engine instances.

**Using the GCP instance pricing calculator**

GCP doesn’t really differ from Azure and AWS. GCP has a full catalog of predefined instances that can be deployed to a GCP project. The E2 instances are the standard machines, while the M-series is specially designed for heavy workloads with in-memory features, running up to and over 1 TB of memory. Details on the GCP catalog can be found at <https://cloud.google.com/compute/vm-instance-pricing>. And obviously, GCP has a calculator too. In the following figure, we’ve ordered a standard E2 instance with a free Linux operating system:



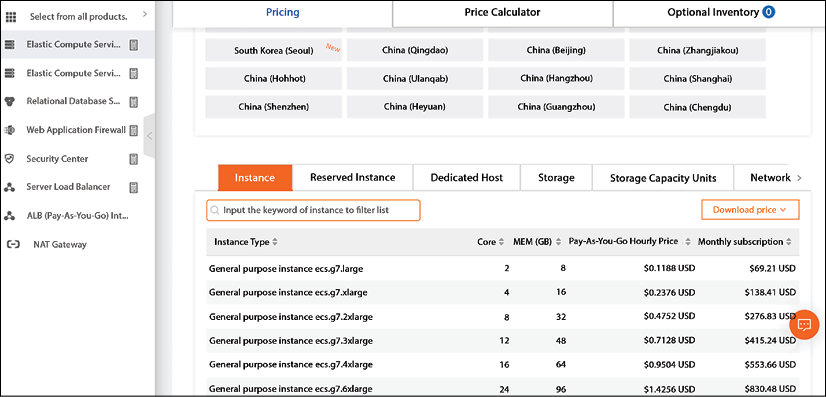
*Figure 10.8: Defining specifications for a VM in the GCP pricing calculator*

The pricing calculator for GCP can be accessed at <https://cloud.google.com/products/calculator>.

**Understanding pricing in Alibaba Cloud**

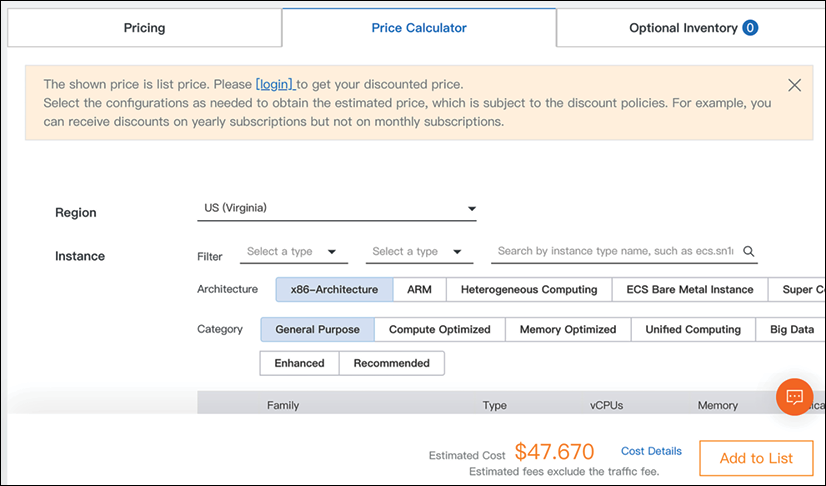
The pricing calculator for Alibaba Cloud can be found at <https://www.alibabacloud.com/pricing-calculator#/commodity/vm_intl>. The first thing that users will notice is that the calculator has three tabs, comparable to the Azure calculator.

The first tab contains the pricing list. The user picks a region at the top and the service for which they want the price tag at the left-hand side of the screen. You can also already choose options for a resource, for instance, pay-as-you-go or reserved instances.



*Figure 10.9: The price list for Alibaba Cloud*

The second tab is the pricing calculator. It will calculate the resources against the list prices. The user must log in to the console to get the discounted prices. Refer to the message at the top of the screen in *Figure 10.10*:

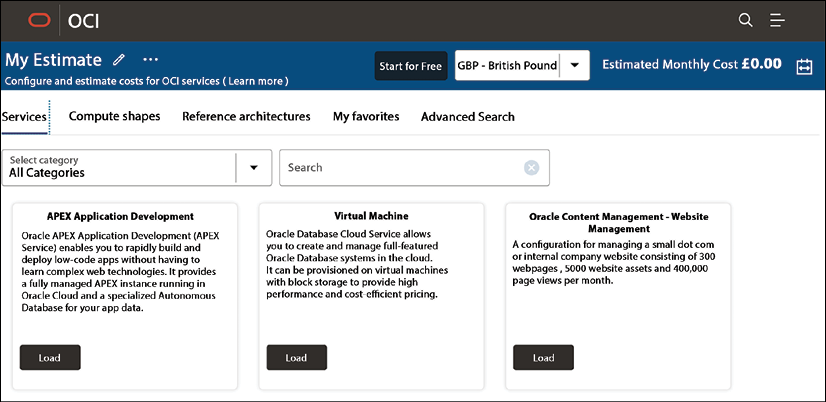


*Figure 10.10: The pricing calculator for Alibaba Cloud*

There’s a third tab, named **Optional Inventory**. This list will populate as soon as you have purchased resources, showing the actual discounts that apply to the account.

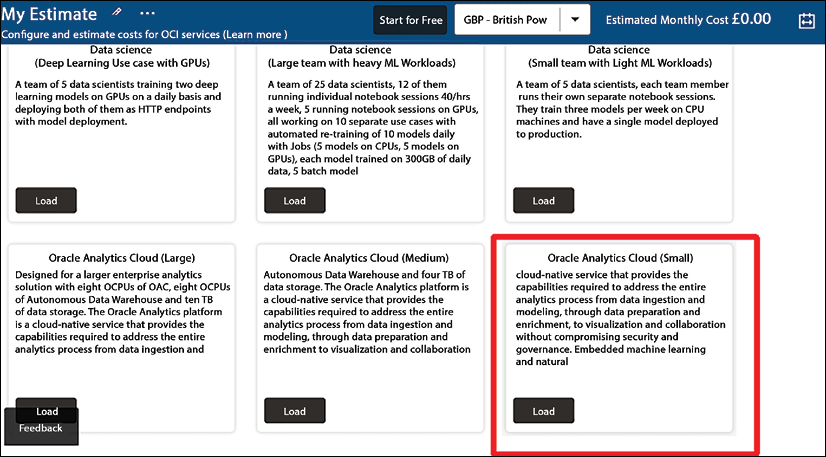
**Using the cost estimator in Oracle Cloud Infrastructure**

Where AWS, Azure, GCP, and Alibaba have one calculator that covers all services for all regions, Oracle offers a cost estimator per region. As an example, you can go to <https://www.oracle.com/uk/cloud/costestimator.html> to find the cost calculator for the UK. It’s shown in *Figure 10.11*:



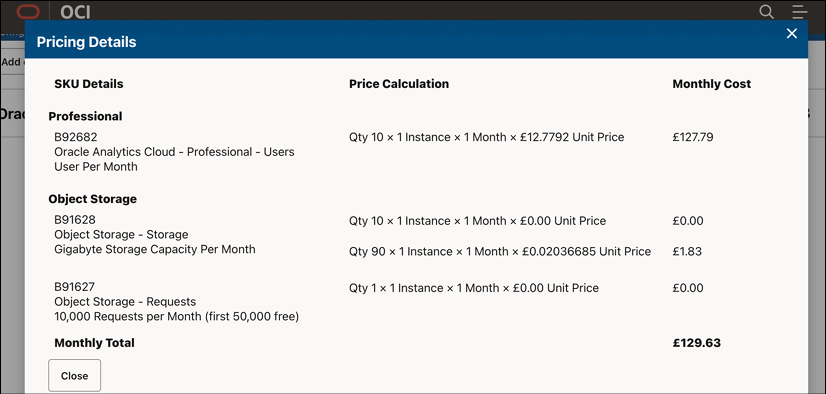
*Figure 10.11: The pricing calculator for OCI*

Under the tab named **Compute shapes**, you will find the various resources and corresponding prices. Reference architectures will help you in setting up specific stacks in OCI, for instance, a small stack to do data analytics. In that case, we can load **Oracle Analytics Cloud (Small)**, as shown in *Figure 10.12* below:



*Figure 10.12: Loading a calculation from the OCI cost estimator*

Now we can view the pricing details, presented in British pounds, shown in the next image:



*Figure 10.13: Example of pricing details in OCI*

Be aware that this only shows the estimated costs for a specific solution or a stack. It does not enroll the configuration to OCI from the estimator.

So far, we have looked at the major cloud platforms and how to purchase and provision VMs to the cloud environments. The next step is to track the costs of resources that we have deployed. One of the first questions that we must answer is from what level we want to track these costs and how we can identify these costs. It’s the topic of the next sections, where we will discuss account hierarchy and tagging as one of the key elements of cost control.

**Understanding account hierarchy**

It’s important to understand from what level enterprise management wants to see costs. Enterprises usually want a full overview of the total spend; hence, we need to make sure that they can view that total spend from the top level all the way down to subscriptions that are owned by specific business divisions or even DevOps teams. These divisions or teams might have a full mandate to run their own subscriptions, but at the top level, the enterprise will want to see the costs that these units are accruing at the end of the day.

This starts with the setup of the tenants, the subscriptions, and the accounts on public cloud platforms. This has to be set up following a specific hierarchy. The good news for financial controllers is that these structures in the public cloud closely follow the rules of the **Chart of Accounts** (**COA**) hierarchy, which is used for financial reporting.

This hierarchy has one top level. There can be many accounts underneath, but at the end of the day, they are all accountable to that top level. There’s no difference when setting up an account hierarchy in the public cloud.

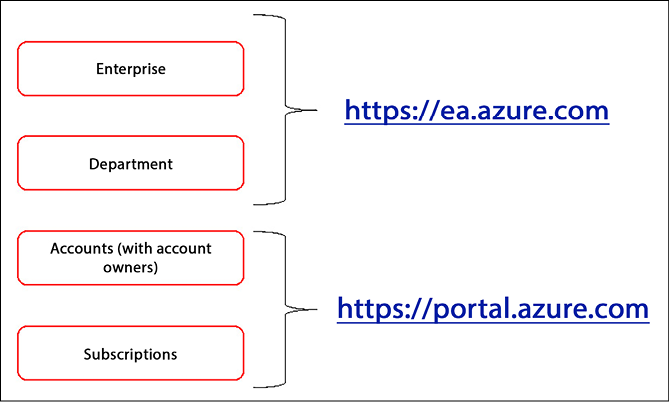
Let’s look at a few examples in Azure, AWS, and GCP.

**Enterprise enrolment in Azure**

In Azure, we work with enterprise enrollment, the top level where we can manage our enterprise administrators and view all usage across all accounts in our Azure environment. The next level is the departments. Beneath the departments, we can create accounts. Both the top level—enterprise enrollment—and the departments are created through the Enterprise Agreement portal at [https://ea.azure.com](https://ea.azure.com/).

Be aware that you need an enterprise account in Azure before you can enter the portal. More information on enterprise enrollment in Azure can be found at <https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/landing-zone/design-area/azure-billing-enterprise-agreement>.

Now we can create accounts. These will be the account owners, who can view all of their subscriptions. The account owners will have the rights to create subscriptions and appoint service administrators that can manage the subscriptions. The following diagram shows the account hierarchy in Azure:



*Figure 10.14: A high-level overview of enterprise enrollment in Azure*

**Organizations in AWS**

In AWS, we can also enroll multiple accounts and centrally manage them. For this, AWS offers a service called AWS Organizations, where we can provision accounts using AWS CloudFormation and group them into organizational units that we can manage. Organizations also allows us to have a centralized cost management platform in AWS.

A service that needs mentioning here is AWS Control Tower, which allows the central management of multi-accounts in AWS, including AWS Organizations, AWS Identity and Access Management, AWS Config, AWS CloudTrail, and AWS Service Catalog. What Control Tower does is launch a landing zone in AWS with automated workflows and blueprints, adding configurations compliant with defined guardrails and setting up a structure to manage the workloads in this landing zone, with the underlying accounts. In short, we could specify Control Tower as a central governance and management dashboard to facilitate and automate enrollments.

To start enterprise enrollment, AWS advises contacting a sales representative directly. This is indeed strongly advised, since AWS has some interesting enterprise volume-driven discount programs such as the**Enterprise Discount Program**(**EDP**), which would be part of an Enterprise Agreement with AWS.

**Organizations in GCP**

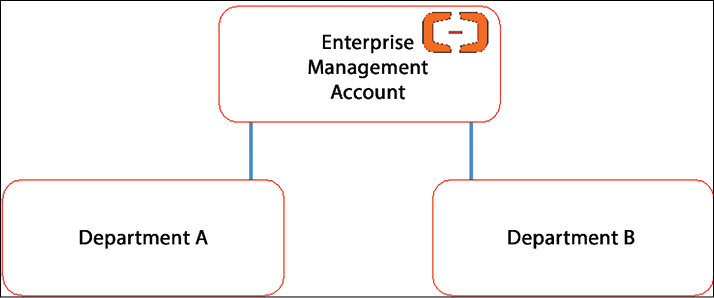
The setup in **GCP** is very similar to Azure. In GCP, we also have a top level, the organization resource. This resource requires an organization node. We create the organization node through Cloud Identity. The node can match the corporate internet domain. Beneath the organization node, we can view and manage every resource and account that is deployed under the organization.

The second level in Azure was departments; in GCP, these are called **folders**. The final layer in the GCP hierarchy is the projects. Projects are functionally similar to subscriptions in Azure. Everything in GCP is created and managed through the Google Cloud console or the gcloud tool. In the console, we create an organization ID. Whoever creates this ID is automatically assigned as the super-administrator.

**Account hierarchy in other clouds**

The principles of account hierarchy in Alibaba and OCI do not differ from the examples that we discussed for Azure, AWS, and GCP. OCI works with compartments to organize and isolate resources. The highest level in OCI is the Administrators group. From this group, engineers have access to compartments. The structure for account hierarchy in Alibaba Cloud looks very similar to Azure, where we will have an enterprise management account at the top with a root folder.

From this account, the departments, as Alibaba refers to isolated environments, can be viewed and managed. The principle is shown in the following diagram.



*Figure 10.15: Simplified root account hierarchy in Alibaba Cloud*

Consolidated billing can be executed from the top, the root enterprise management account.

This concludes the section about understanding the hierarchy of enterprise organizations in the major public clouds. The main conclusion is that the models resemble one another very closely, for a very good reason: at the top level, it is best for an enterprise administrator to have one single view of everything that is deployed in a cloud environment.

**Understanding license agreements**

License agreements are complicated, but in essence, there are three types of agreements to start using services in the public cloud:

* **Consumption-based**: This is often referred to as the pay-as-you-go model. The enterprise only pays for the actual usage in the public cloud, without any up-front commitment. Cloud providers issue a monthly invoice with the actual consumption of resources. These resources—for example, VMs, database instances, and storage units—are charged against the rates that are published on the public portals of the providers.
* **Commitment-based**: For most enterprises, this is the preferred model. In this case, the enterprise commits to the usage of a specific amount of resources in the cloud for a longer period of time, typically 1, 3, or 5 years. Now, public clouds such as Azure, AWS, and GCP were invented to enable maximum flexibility and agility.

If we allow enterprises to commit resources for a longer period, then this will have an impact on the resources that a public cloud can offer to other customers. For that reason, public cloud providers want to be certain that enterprises do really commit to the consumption of these reserved resources. Typically, an enterprise will need to pay upfront for these resources, whether they use them or not. Cloud consumption has become a formal contract that entitles an enterprise to have these resources available at all times.

* **Limited agreements**: These are agreements that are limited by time to an amount of resources that a customer can use. Typically, these are the type of agreements that are used for trial periods where resources are not charged for a specific period. Not all services will be part of these agreements—such as heavy instances with a lot of memory and terabytes of storage—and after 1, 2, or 3 months, the environment will be suspended by the cloud provider. A limited agreement can also hold a certain number of credits that can be used for a given time. If the credit amount is used, the trial period ends.

We need licenses to get started. Next, we can start provisioning resources. How do we identify and track these resources? Tags can be of great help. We will discuss this in the next section.

**Define tagging standards**

The major benefit of cloud provisioning is that an organization doesn’t need to make large investments in on-premises infrastructure. In the public cloud, it can deploy and scale resources whenever needed and pay for these resources as long as the organization uses them. If it doesn’t use the resources, it will not receive an invoice—unless a company has contracted reserved instances.

Another advantage of cloud provisioning is the agility and speed of deployment. Developers can easily deploy resources within a few minutes. But that’s a budget risk at the same time. With on-premises investments, a company knows exactly what the costs will be over a certain period: the investment itself and the depreciation are a given. The cloud works differently, but an organization needs to be able to forecast the costs and control them.

A way to do this is by tagging resources. Tags allow a company to organize the resources in its cloud environment in a logical way, so they can easily be identified. By grouping resources using tags, it’s also easy to see what costs are related to these resources and to which department or company division these costs should be transferred.

Tags are likely the most important attribute in terms of cost management and control in cloud environments. Naming conventions are much more focused on the identification of resources and are also crucial to the automation of cloud management. Tags are metadata that allow additional information on resources that can’t be stored in a name. Tagging helps in understanding cost allocation, since we can use tags to categorize cloud resources.

All cloud providers offer extensive ways to apply tags to our resources. However, standard tags can be utilized across these different clouds. It’s recommended to have tags for at least the following attributes:

* **Application**: Typically, a resource is part of an application or an application stack. To categorize resources—meaning VMs, storage, databases, and network components—that belong to one application or application stack, a tag should be added to identify to what stack or application a resource belongs.
* **Billing**: Especially large enterprises will have divisions, business units, or brands. These entities might have budgets or might be separate cost centers. Tags will ensure that resources are billed to specific budgets, or the accounting cost centers.
* **Service class**: Tags can indicate what service level is applicable to resources. Are they managed 24/7, what is the patch schedule, and what is the backup scheme? Often, enterprises have a tiered categorization for resources, such as gold, silver, and bronze. Gold is the highest level for production systems and may have disaster recovery solutions and uptime of 99.999%; silver and bronze would be for single systems with a much lower service level. A tag indicating gold, silver, or bronze will make clear what the service class of that particular resource is.
* **Compliance**: These tags indicate whether compliance rules apply. These can be industry compliance regulations, such as for healthcare or financial institutions, as well as internal compliance rules. These can be important in, for example, granting access to specific resources or the way data is securely stored.

Tags are a must to identify resources and the costs these resources generate. They will help us in validating invoices, which is the topic of the next section.

**Validate and manage billing**

It’s very likely that a multi-cloud strategy will place several migrated systems into multiple different public clouds. With that, we are generating costs for pay-per-use instances and services, reserved instances for which companies have longer-term obligations, and licenses. Invoices will arrive from different providers. How do we keep track of all that?

Let’s have a look first at billing in the major cloud platforms being discussed in this book: Azure, AWS, GCP, Alibaba Cloud, and OCI. These platforms share the same billing approach: as soon as services are consumed on the platform, charges will begin to accrue to which the CSPs can send invoices. Typically, this is referred to as the billing account. We will be using the cost or billing dashboards of the clouds to view costs and invoices.

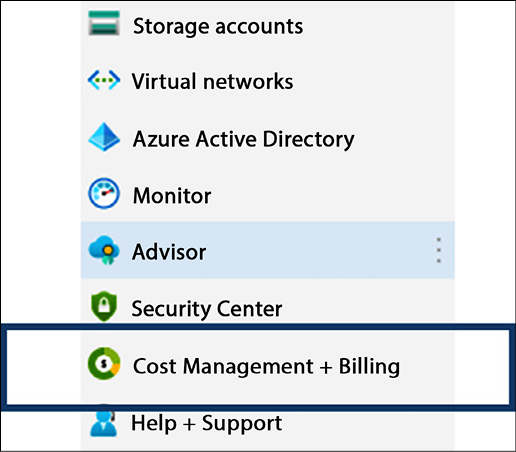
**Using cost management and billing in Azure**

Azure billing has three types of billing:

* **Microsoft Online Services program**: Every user in Azure starts in this program. As soon as you sign in to Azure through the portal, you will get a billing account. This is also the case when you sign up for a free account. It’s also needed for all pay-as-you-go services and for a subscription to Visual Studio, the Microsoft tool for development in cloud environments.
* **Enterprise Agreement**: An organization can sign an **Enterprise Agreement** (**EA**) with Microsoft to use Azure, which is valid across a lot of other products and services of Microsoft. An EA is a monetary commitment. An organization is entitled to extensive support from Microsoft, but it comes with contractual obligations such as a minimum spend.
* **Microsoft Customer Agreement**: If an organization signs up for Azure, in most cases a billing account will be issued for a Microsoft Customer Agreement. In some Azure regions, a Microsoft Customer Agreement can be issued if a free account is upgraded.

When a billing account is activated, a billing profile will be attached to it. This profile enables managing invoices and payments. Azure creates monthly invoices at the beginning of each month. Depending on the billing profile, the person who owns the account will see all costs associated with subscriptions and services in those subscriptions that are purchased under that specific account.

For example, if the billing profile is set to enterprise level, the billing account lists all costs that a company generates in Azure, in all subscriptions within the enterprise tenant. It’s advised to define more billing profiles with specific invoice sections. This is done in the Azure portal under the **Cost Management + Billing** option, as shown in the following screenshot:



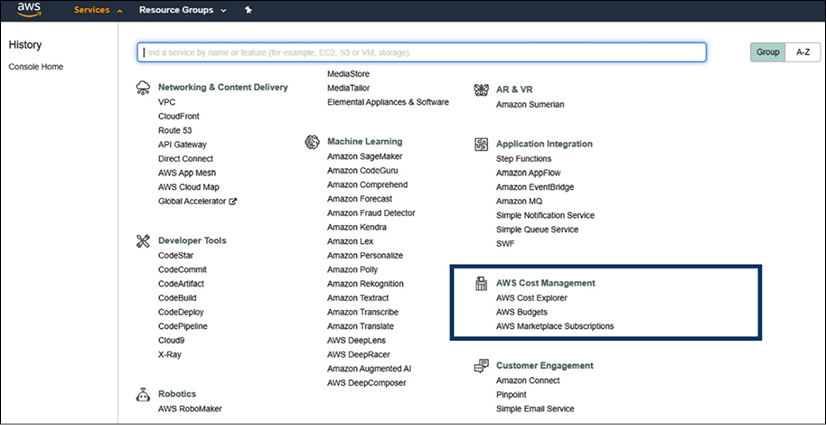
*Figure 10.16: The Cost Management + Billing option in the Azure portal menu*

It’s true that the overall concept of cost management and billing is pretty much the same in the different clouds, but there are some differences in possible implementations for our own organization. Next, we will take a look at AWS and GCP.

**Using AWS Cost Management for billing**

In AWS, the free tier is the typical entry point and provides a lot of services for us to use. Organizations will typically enter into a customer agreement with AWS. Be aware that if you sign up on behalf of a company, AWS considers you the person with the legal authority to do so. Make sure that you are entitled to get into a commitment on behalf of your organization.

Similarly to the way Azure sets it up, cost and billing management for AWS is viewed through its portal. It’s under the **AWS Cost Management** menu item, as shown in the following screenshot:



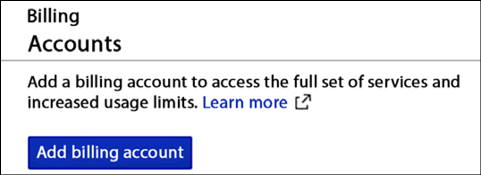
*Figure 10.17: Cost and billing in the AWS console menu*

It’s common in both AWS and Azure for an organization to have separate divisions of accounts. With AWS Organizations, consolidating billing can be activated. There’s one account for the whole organization or multiple accounts reflecting the organizational structure of the company. In the latter case, there will be multiple accounts. These accounts can be grouped under one, consolidated master account to have an overview of all AWS costs generated.

In AWS Cost Management, we can analyze costs with Cost Explorer, get usage reports, and manage our payments. Billing preferences can be set, such as receiving billing alerts and invoices being emailed as PDFs. Payment preferences, such as paying through a credit card or bank account, can also be set in Cost Management. In Europe, the **Single Euro Payments Area** (**SEPA**) is commonly used. In India, payments can be submitted through **Amazon Internet Services Private Limited** (**AISPL**).

**Using billing options in GCP**

As soon as billing is activated in GCP, the portal will prompt the user to set a billing account, as shown in the following screenshot:



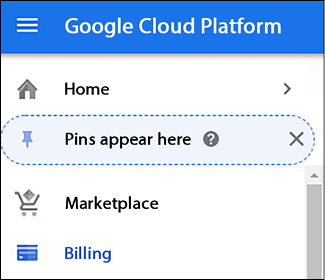
*Figure 10.18: Adding a billing account in GCP*

In GCP, cloud billing accounts are always associated with projects, which are the equivalent of subscriptions in Azure and accounts in AWS. Like in Azure, the billing account is coupled with a Google payments profile. There are two types of billing account roles:

* **Billing account admin**: This is typically someone in the finance department. This account can view all costs, set budgets and billing alerts, and link or unlink projects.
* **Billing account user**: The only thing the user can do is link a project to a billing account and see the costs associated with that project. The user can’t unlink the project, unlike the admin.

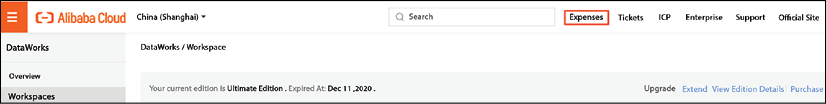
The payment profile contains information about the legal entity that is responsible for the accounts. It also stores information on tax obligations such as VAT, bank accounts, payment methods, and transaction information such as outstanding invoices. Only the billing account admin role can view and alter this information.

In the Google Cloud console, we can enable interactive billing reports where the views and reports on billing information can be customized. For example, cost breakdowns can be added per project or per service used in GCP. In the Google Cloud console, this is all featured under **Billing** in the main menu, as shown in the following screenshot:



*Figure 10.19: The Billing tab in Google Cloud’s console*

To view costs in Alibaba Cloud, we need to go to the Alibaba Cloud Management Console and browse to the **Expenses** tab:



*Figure 10.20: The Expenses tab in the Alibaba Cloud Management Console*

Before we do that, we must ensure that a role has specific access to the billing information—no different from the other clouds. In Alibaba, we need to grant access to the **Resource Access Management** (**RAM**) user. In fact, this RAM user must have an AliyunBSSFullAccess policy attached to the role. Only then will the user be able to view the billing information. On the billing page, we can then select **Spending Summary**, which will show the expenses per period or per subscription.

It’s no different in OCI: to enable users to view and track costs in an OCI account, the user will need access provided through a policy. With the appropriate access, the user can manage payments and budgets, view cost and usage reports, and also do cost analysis. Costs can be filtered through dates, compartments (remember that isolated environments in OCI are called compartments), and/or tags.

In this section, we have explored the various billing options in Azure, AWS, GCP, Alibaba, and OCI using the billing dashboards in these clouds. In the next section, we will learn how we can validate invoices.

**Validating invoices**

Validating invoices has nothing to do with checking whether a cloud provider has charged us the correct amounts. Cloud providers have fully automated this process, so you may rest assured that if you or your company uses a resource in their cloud, it will show up on the bill. Validating invoices is about checking whether invoiced costs correspond with the forecasted usage of your company. Are you on budget or are you overspending? Are there resources on the bill that you aren’t using anymore? And if so, why didn’t you delete these resources?

Some key decisions will need to be made. These decisions are the same for all clouds covered:

* Will the organization use one or multiple billing accounts? If you want a project manager to be able to validate the costs for a specific project or in a particular environment, then they should be granted access to view these costs. As we have seen in the previous section, we can set these privileges granularly in roles and profiles that are attached to billing accounts.
* How will payments be processed? As discussed in the previous section, cloud providers offer various ways to process payments. Credit cards are popular, but most enterprises do their payments through invoiced billing and their respective bank accounts. The latter is strongly recommended for optimized cost control.

Next, we must define the validation process. It might sound overdone, but the truth is that organizations tend to have significant overspending in public clouds—simply because they lack insights and control into the billing process and because they don’t have accurate cost management in place.

A recommended approach comes in three steps:

1. **Project control**: A project manager, product owner, or Scrum Master should be aware of what costs are generated from a project. If a team works in Agile Scrum and uses Sprints, it is advised to validate deployed resources after each Sprint. Are the designed resources deployed and what other services are related to that? These overviews should match the costs that are allocated to the project.
2. **Architecture control**: The role of the architect is to verify that only resources that are agreed-upon artifacts are included. A simple example: if it is agreed that only VMs of a certain series may be used for deployment to production, then the architect should check that this requirement is met. The deployment of other resources could inflict higher costs.
3. **Finance and accounting**: Based on the checks by project management and architecture control, the finance department can be sure that resource deployment is done correctly and that costs can be accounted for. Finance now has the task of checking invoices on terms of payment conditions and contractual agreements.

All cloud providers have dashboards that show exactly how much resource consumption will cost. That’s a reactive approach, which can be fine. But if we want to force teams and developers to stay within budgets, we can set credit caps on subscriptions and have alerts raised as soon the cap is reached. All discussed cloud providers offer services to set budgets and alerts from the billing or expenses pages in their management consoles.

In multi-cloud, this would however imply that we need to view the dashboards of the various cloud providers. Enterprises would likely want to have a single pane of glass view: a central dashboard showing all the financial data of the total cloud consumption. One tool that can help with that is CloudHealth by VMware, which provides insights and advice for cloud financial management across multiple clouds. CloudHealth is a member of the FinOps Foundation, and hence follows the principles of the foundation.

**Summary**

This chapter started with a brief overview of the principles for FinOps: financial operations in the cloud or cloud financial management. We studied how we can provision resources for various clouds and then learned how we can track costs that are related to these resources. Before we can track resources, view the associated costs, and validate invoices, we must understand how cost management works in the cloud. We discussed the cost tools in Azure, AWS, GCP, Alibaba Cloud, and OCI. All these providers offer comprehensive toolsets to provision and identify resources from their respective management consoles. However, we must understand some general principles, such as license agreements and tagging.

In this chapter, we discussed the foundation of FinOps. In the next chapter, we will elaborate on how organizations can implement and develop cloud financial management, including the setup of a FinOps practice, using the FinOps maturity model.

**Questions**

1. If we want to run a trial period in a public cloud, what type of agreement would fit our needs?
2. Cloud providers use different technology to provision resources. What technology do both Alibaba Cloud and OCI use?
3. What is the discount program for large accounts in AWS called?
4. True or false: the pricing calculators of cloud providers are free to use.

**Further reading**

* *The Road to Azure Cost Governance*, by Paola E. Annis and Giuliano Caglio, 2022, Packt Publishing
* *AWS FinOps Simplified*, by Peter Chung, 2022, Packt Publishing