**6**

**Controlling the Foundation Using Well-Architected Frameworks**

This chapter of the book is all about the basic operations, or BaseOps for short, in multi-cloud. We’re going to explore the foundational concepts of the major cloud providers and learn about the basics, starting with managing the landing zone—the foundation of any cloud environment. Before a business can start migrating workloads or developing applications in cloud environments, they will need to define that foundation. Best practices for landing zones include using enterprise-scale landing zones in Azure, AWS Landing Zone, and landing zones and projects in Google Cloud and Alibaba Cloud.

In this chapter, we will cover the following topics:

* Understanding BaseOps and the foundational concepts
* Building the landing zone with Well-Architected and cloud adoption principles
* BaseOps architecture patterns
* Managing the landing zone using policies
* Understanding the need for demarcation

**Understanding BaseOps and the foundational concepts**

**BaseOps** might not be a familiar term to all, although it is simple to interpret: **basic operations**. In cloud environments, these are more often referred to as **cloud operations**. BaseOps is mainly about operating the cloud environment in the most efficient way possible by making optimal use of the cloud services that major providers offer on the different layers: network, compute, storage, but also PaaS and SaaS.

The main objective of BaseOps is to ensure that cloud systems are available to the organization and that these can safely be used to do the following:

* Monitor network capacity and appropriately route traffic.
* Monitor the capacity of compute resources and adjust this to the business requirements.
* Monitor the capacity of storage resources and adjust this to the business requirements.
* Monitor the availability of resources, including health checks for backups and ensuring that systems can be recovered when required.
* Monitor the perimeter and internal security of systems, ensuring data integrity.
* Overall, manage systems at the agreed-upon service levels and use **Key Performance Indicators** (**KPIs**), as agreed upon by the business.
* Assuming that the systems are automated as much as possible, part of BaseOps is also being able to monitor and manage the pipeline.

At the end of the day, this is all about the quality of service. That quality is defined by service levels and KPIs that have been derived from the business goals. BaseOps must be enabled to deliver that quality via clear procedures, skilled people, and the proper tools.

We have already explored the business reasons regarding why we should deploy systems in cloud environments: the goal is to have flexibility, agility, but also cost efficiency. This can only be achieved if we standardize and automate. All repetitive tasks should be automated. Identifying these tasks and monitoring whether these automated tasks are executed in the right way is part of BaseOps. The automation process itself is development, but a key reason we should have DevOps in the first place is so that we can execute whatever the developer invents. Both teams have the same goal, for that matter: protect and manage the cloud systems according to best practices.

We can achieve these goals by executing the activities mentioned in the following sections. The Well-Architected Frameworks are a good methodology to capture the requirements of a cloud implementation.

**Defining and implementing the base infrastructure—the landing zone**

This is by far the most important activity in the BaseOps domain. It’s really the foundation of everything else. The landing zone is the environment on the designated cloud platform where we will host the workloads, the applications, and the data resources. The starting principle of creating a landing zone is that it’s fully provisioned through code. In other words, the landing zone contains the building blocks that form a consistent environment where we can start deploying application and data functionality, as we discussed in *Chapter 4*, *Service Designs for Multi-Cloud*, where we talked about scaffolding.

First, let’s define what a landing zone is. Landing zones are the place to start building environments in public clouds. Let’s use this analogy: if we want to land an aircraft, we need an airfield. The airfield is much more than just a long strip of tarmac where the plane actually lands. We also need some form of governance such as air traffic controllers who guide the plane toward the landing strip and decide in what order planes can land to keep it all safe.

That’s exactly what landing zones in the cloud are: a place to land workloads. As we discussed in the previous chapter, the public cloud providers have **Cloud Adoption Frameworks** (**CAFs**) and **Well-Architected Frameworks** (**WAFs**) to help build the right landing zone, based on the requirements of the workloads that are supposed to land in the public cloud.

We could also compare landing zones to the foundations of a house. An architect must know how the house will look before an engineer can start calculating how strong and what shape the foundations must be on which the house stands. This implies that we must know what sort of workloads we will host on the landing zone in the public cloud before we start designing and building the landing zone. The architect must at a minimum have an idea of what the cloud platform will be used for in terms of applications and type of data. There have to be business requirements to create the right landing zone. These requirements are translated into standards and guardrails.

**Defining standards and guardrails for the base infrastructure**

The base infrastructure typically consists of networking and environments that can host, compute, and store resources. You could compare this with the **Hyperconverged Infrastructure** (**HCI**), which refers to a physical box that holds compute nodes, a storage device, and a switch to make sure that compute nodes and storage can communicate. The only addition that we would need is a router that allows the box to communicate with the outside world. The cloud is no different: the base infrastructure consists of compute, storage nodes, and switches to enable traffic. The major difference with the physical box is that, in the cloud, all these components are code.

But as we have already learned, this wouldn’t be enough to get started. We also need an area that allows us to communicate from our cloud to the outside and to access our cloud. Next, we will need to control who accesses our cloud environment. So, a base infrastructure will need accounts and a way to provision these accounts in a secure manner. You’ve guessed it: even when it comes to defining the standard and policies for setting up a base infrastructure, there are a million choices to make. Landing zone concepts make it a lot easier to get started fast.

As a rule of thumb, the base infrastructure consists of five elements:

* Network
* Compute nodes
* Storage nodes
* Accounts
* Defense (security)

The good news is that all cloud providers agree that these are the base elements of an infrastructure. Even better, they all provide code-based components to create the base infrastructure. From this point onward, we will call these components building blocks. The issue is that they offer lots of choices in terms of the different types of building blocks and how to deploy them, such as through blueprints, templates, code editors, command-line programming, or their respective portals.

The Well-Architected Frameworks provide good guidance on making the right choices.

**Building the landing zone with Well-Architected and Cloud Adoption principles**

AWS explains it well on their website: the landing zone is a Well-Architected environment that is scalable and secure. The Well-Architected Framework provides prescriptive guidance on how to design and build the landing zone. In *Chapter 4*,*Service Designs for Multi-Cloud*, we discussed the various pillars of the framework. In this section, we will see how the guidelines are used in designing and building the landing zones.

All cloud providers offer services that enable the design and implementation of landing zones, using the principles of the Cloud Adoption and Well-Architected Frameworks. The terms might change per cloud provider, but the overall principles are all the same in Azure, AWS, GCP, and Alibaba. Having said that, the commercial offerings are different per provider. We will discuss the propositions per provider.

**Enterprise-scale in Azure**

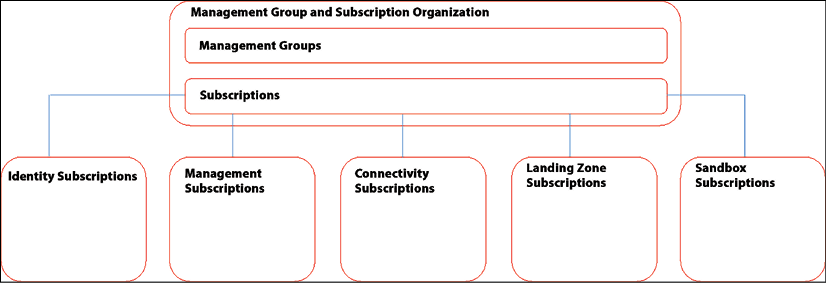
Microsoft calls the landing zone the “plumbing” that must be in place before customers can start hosting workloads in Azure. From the CAF, the landing zone contains best practices for scalability, governance, security, networking, and identity management. In 2020, Microsoft introduced enterprise-scale landing zones. This is a service that offers prescriptive architectures and design principles. Enterprise-scale delivers a ready-to-go landing zone for eight architecture domains: Microsoft themselves call it an “accelerator.” Landing zones provide many implementation options built around a set of common design areas. So, it’s an actionable resource. You can deploy the templates provided by Microsoft:

* **Enterprise Agreement** (**EA**) enrollment and Azure Active Directory tenants
* Identity and access management
* Management group and subscription organization
* Network topology and connectivity
* Management and monitoring
* Business continuity and disaster recovery
* Security, governance, and compliance
* Platform automation and DevOps

Standardized Microsoft Azure offerings and services are included in these architectures, including **Privileged Identity Management** (**PIM**), web application firewalls, and Azure Virtual WAN for connectivity. Enterprise-scale sets up a number of subscriptions in Azure that provide all the foundational services to start hosting workloads on the platform:

* Identity subscription with Azure Key Vault
* Management subscription for monitoring and logging
* Connectivity subscription with Virtual WAN, and hub-and-spoke models
* Sandbox subscription for testing applications and services
* Landing zone subscription for production workloads with applications

These are all part of the subscription with the root management group and subscription organization. The following diagram shows a high-level representation of an enterprise-scale landing zone:



*Figure 6.1: High-level representation of an enterprise-scale landing zone in Azure*

In the enterprise-scale landing zone, there is an area for “landing zone subscriptions.” Those are the ones for applications or workloads.

More information about enterprise-scale landing zones in Azure can be found at <https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/landing-zone/>.

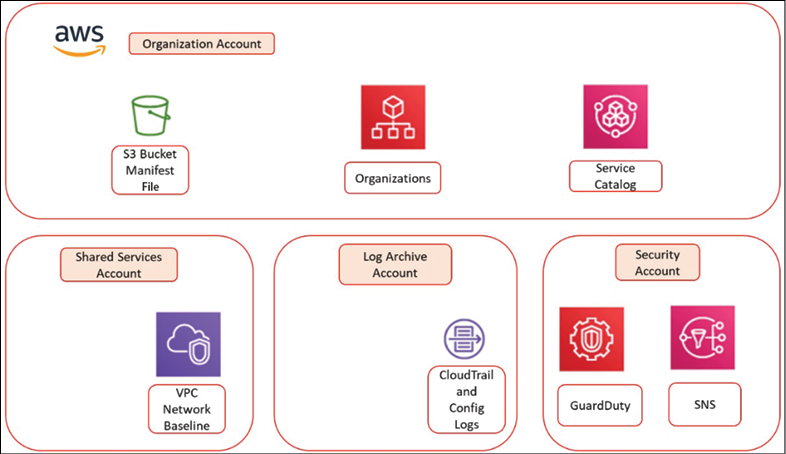
**AWS Landing Zone**

AWS offers AWS Landing Zone as a complete solution, based on the Node.js runtime. Like Azure, AWS offers numerous solutions so that you can set up an environment. All these solutions require design decisions. To save time in getting started, AWS Landing Zone sets up a basic configuration that’s ready to go. To enable this, AWS Landing Zone deploys the so-called AWS **Account Vending Machine** (**AVM**), which provisions and configures new accounts with the use of single sign-on.

AWS uses accounts. AWS Landing Zone comprises four accounts that follow the CAF of AWS:

* **Organization account**: This is the account that’s used to control the member accounts and configurations of the landing zone. It also includes the so-called manifest file in the S3 storage bucket. The manifest file sets parameters for region and organizational policies. The file refers to AWS CloudFormation, a service that we could compare to ARM in Azure. CloudFormation helps with creating, deploying, and managing resources in AWS, such as EC2 compute instances and Amazon databases. It supports Infrastructure as Code.
* **Shared services account**: By default, Landing Zone manages the associated accounts through **SSO**, short for **single sign-on**. The SSO integration and the AWS-managed AD are hosted in the shared services account. It automatically peers new accounts in the VPC where the landing zone is created. AVM plays a big role in this.
* **Log archive account**: AWS Landing Zone uses CloudTrail and Config logs. CloudTrail monitors and logs account activity in the AWS environment that we create. It essentially keeps a history of all actions that take place in the infrastructure that is deployed in a VPC. It differs from CloudWatch in that it’s complementary to CloudTrail. CloudWatch monitors all resources and applications in AWS environments, whereas CloudTrail tracks activity in accounts and logs these activities in an S3 storage bucket.
* **Security account**: This account holds the key vault—the directory where we store our accounts—for cross-account roles in Landing Zone and the two security services that AWS provides: GuardDuty and Amazon SNS. GuardDuty is the AWS service for threat detection, the **Simple Notification Service** (**SNS**) that enables the sending of security notifications. Landing Zone implements an initial security baseline that comprises (among other things) central storage of config files, the configuration of IAM password policies, threat detection, and Landing Zone notifications. For the latter, CloudWatch is used to send out alerts in case of, for example, root account login and failed console sign-in.

The following diagram shows the setup of the landing zone in AWS:



*Figure 6.2: The AWS Landing Zone solution*

The one thing that we haven’t covered in detail yet is AVM, which plays a crucial role in setting up Landing Zone. AVM launches basic accounts in Landing Zone with a predefined network and security baseline. Under the hood, AVM uses Node.js templates that set up organization units wherein the previously described accounts are deployed with default, preconfigured settings. One of the components that is launched is the AWS SSO directory, which allows federated access to AWS accounts.

More information about AWS Landing Zone can be found at [https://aws.amazon.com/solutions/aws-landing-zone/.](https://aws.amazon.com/solutions/aws-landing-zone/)

**Landing zones in Google Cloud**

GCP differs a lot from Azure and AWS, although the hub-and-spoke model can also be applied in GCP. Still, you can tell that this platform has a different vision of the cloud. GCP focuses more on containers than on IaaS by using more traditional resources. Google talks about a landing zone as somewhere you are planning to deploy a Kubernetes cluster in a GCP project using GKE, although deploying VMs is, of course, possible on the platform.

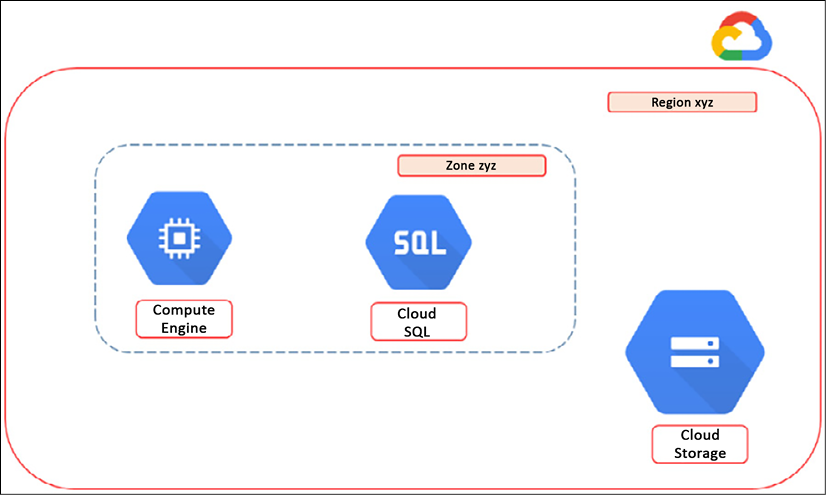
In the landing zone, you create a **Virtual Private Cloud** (**VPC**) network and set Kubernetes network policies. These policies define how we will be using isolated and non-isolated Pods in our Kubernetes environment. Basically, by adding network policies, we create isolated Pods, meaning that these Pods—which hold several containers—only allow defined traffic, whereas non-isolated Pods accept all traffic from any source. The policy lets you assign IP blocks and deny/allow rules to the Pods. The next step is to define service definitions to the Kubernetes environment in the landing zone so that Pods can start running applications or databases. The last step to create the landing zone is to configure DNS for GKE.

As we mentioned previously, Google very much advocates the use of Kubernetes and containers, which is why GCP is really optimized for running this kind of infrastructure. If we don’t want to use container technology, then we will have to create a project in GCP ourselves. The preferred way to do this is through Deployment Manager and the gcloud command line. You could compare Deployment Manager to ARM in Azure: it uses the APIs of other GCP services to create and manage resources on the platform. One way to access this is through Cloud Shell within the Google Cloud portal, but GCP also offers some nice tools to get the work done. People who are still familiar with Unix command-line programming will find this recognizable and easy to work with.

The first step is enabling these APIs; that is, the Compute Engine API and the Deployment Manager API. By installing the Cloud SDK, we get a command-line tool called gcloud that interfaces with Deployment Manager. Now that we have gcloud running, we can simply start a project with the gcloud config set project command, followed by the name or ID of the project itself; for example, gcloud config set project [Project ID]. Next, we must set the region where we will be deploying our resources. It uses the very same command; that is, gcloud config set compute/region, followed by the region ID; that is, gcloud config set compute/region [region].

With that, we’re done! Well, almost. You can also clone samples from the Deployment Manager GitHub repository. This repository also contains good documentation on how to use these samples.

The following diagram shows a basic setup for a GCP project:



*Figure 6.3: Basic setup of a project in GCP, using Compute Engine and Cloud SQL*

To clone the GitHub repository for Deployment Manager in your own project, use the git clone command <https://github.com/GoogleCloudPlatform/deploymentmanager-samples> or go to <https://github.com/terraform-google-modules/terraform-google-migrate>. There are more options, but these are the commonly used ways to do this.

**Landing Zone in Alibaba Cloud**

Like AWS and Azure, Alibaba Cloud provides a full service to design and implement a landing zone. It does work a bit differently though: with Alibaba, a customer applies for the service fifteen days before the order is placed. These fifteen days are used to agree on objectives and service requirements. Alibaba collects information that is required to design the solution, including an assessment of the current IT landscape of the customer and the objectives that the customer has to migrate workloads to the cloud.

After the agreement is settled, the customer pays for the solution design and implementation. Alibaba Cloud offers three flavors: basic, standard, and advanced. The pricing is fixed. Basic is charged at 75,000 USD, standard at 150.000 USD, and advanced is based on the business scenario. Prices are for one-off payments.

The Landing Zone service includes account management, financial management, network planning, security management, resource management, and compliance auditing. The latter, compliance auditing, is only provided in standard and advanced.

The landing zone is only implemented after the acceptance of the solution by the customer.

More information about applying for the Landing Zone service in Alibaba Cloud can be found at <https://www.alibabacloud.com/services/alibaba-cloud-landing-zone>.

With that, we have created landing zones in three major cloud platforms and by doing so, we have discovered that, in some ways, the cloud concepts are similar, but that there are also some major differences in the underlying technology. Now, let’s explore how we can manage these landing zones using policies, as well as how to orchestrate these policies over the different platforms.

**Landing zones in Oracle Cloud Infrastructure**

A landing zone in OCI is a pre-defined, automated infrastructure deployment that helps organizations quickly set up a secure, scalable, and efficient cloud environment. Like the other concepts that we have discussed, it’s essentially a best-practice blueprint for cloud infrastructure that includes account structure, networking, security, and governance policies. Using a landing zone can help organizations avoid common pitfalls and accelerate their cloud adoption journey.

Once we have defined our requirements, we can create a landing zone environment using OCI’s Landing Zone service. This service provides a wizard-driven interface that will guide us through the process of creating a landing zone. The service offers choices in various landing zone patterns, such as a standard landing zone, a multi-account landing zone, or a regulated landing zone.

After creating the landing zone environment, we can customize it to meet specific needs. This includes configuring **Virtual Cloud Networks**(**VCNs**), subnets, security controls, and IAM policies. We can also define governance policies and automation using tools such as Terraform.

OCI provides various monitoring and management tools, such as the OCI console, APIs, and a CLI, as well as third-party tools such as Terraform, Ansible, and Chef.

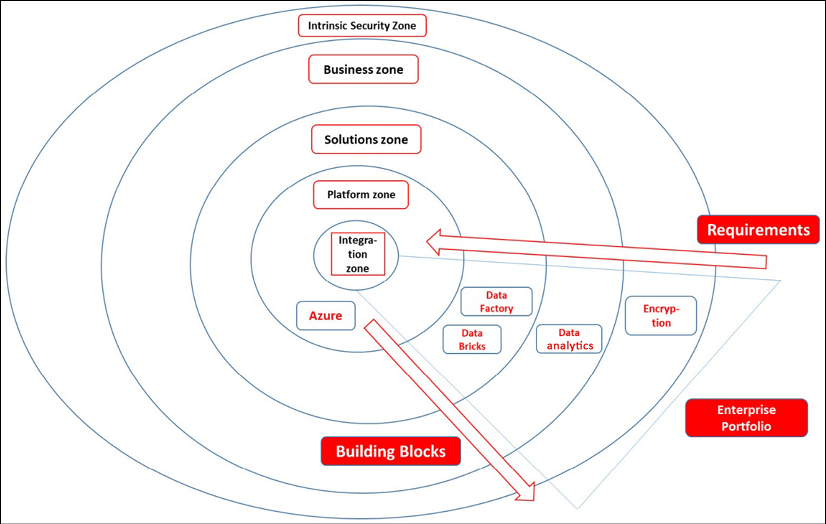
**BaseOps architecture patterns**

A way to define a reference architecture for your business is to think outside-in. Think of an architecture in terms of the circles that can be seen in *Figure 6.4*. The circle just within the intrinsic security zone is the business zone, where all the business requirements and principles are gathered. These drive the next inner circle: the solutions zone. This is the zone where we define our solutions portfolio. For example, if the business has a demand for analyzing large sets of data (a business requirement), then a data lake could be a good solution.

The solution zone is embedded between the business zone on the outer side and the platform zone on the inner side. If we have, for instance, Azure as our defined platform, then we could have Azure Data Factory as part of a data lake solution. The principle is that from these platforms, which can also be third-party PaaS and SaaS platforms, the solutions are mapped to the business requirements. By doing so, we create the solutions portfolio, which contains specific building blocks that make up each solution.

The heart of this model—the most inner circle—is the integration zone, from where we manage the entire ecosystem in the other, outer circles.

Security should be included in every single layer or circle. Due to this, the boundary of the whole model is set by the intrinsic security zone:



*Figure 6.4: Circular model showing the layers of the enterprise portfolio*

The preceding diagram shows this model with an example of a business requiring data analytics, with Data Factory and Databricks as solutions coming from Azure as the envisioned platform. The full scope forms the enterprise portfolio.

This will define the pattern for the architecture, working from the outside in. Next, we can start detailing the foundation architecture with patterns for infrastructure in the cloud. The basics are the same for every public cloud.

For a network, we will have to manage, at a minimum, the following:

* Provisioning, configuring, and managing virtual networks (vNets, VPCs, subnets, internet-facing public zones, and private zones)
* Provisioning and managing routing, **Network Address Translation** (**NAT**), **Network Access Control** (**NAC**), **Access Control Lists** (**ACLs**), and traffic management
* Provisioning and managing load balancing, network peering, and network gateways for VPNs or dedicated connections
* Provisioning and managing DNS
* Network monitoring
* Detecting, investigating, and resolving incidents related to network functions

For compute, we will have to manage, at a minimum, the following:

* Provisioning, configuring, and the operations of virtual machines. This often includes managing the operating system (Windows, various Linux distributions, and so on)
* Detect, investigate, and resolve incidents related to the functions of virtual machines
* Patch management
* Operations of backups (full, iterative, and snapshots)
* Monitoring, logging, health checks, and proactive checks/maintenance

Do note that compute in the cloud involves more than virtual machines. It also includes things such as containers, container orchestration, functions, and serverless computing. However, in the landing zone, these native services are often not immediately deployed. You might consider having the container platform deployed as part of the base infrastructure. Remember that, in the cloud, we do see a strong shift from VMs to containers, so we should prepare for that while setting up our landing zone.

In most cases, this will include setting up a Kubernetes cluster. In Azure, this is done through **Azure Kubernetes Service** (**AKS**), where we create a resource group that will host the AKS cluster. AWS offers its own cluster service through **Elastic Kubernetes Service** (**EKS**). In GCP, this is **Google Kubernetes Engine** (**GKE**). The good news is that a lot of essential building blocks, such as Kubernetes DNS, are already deployed as part of setting up the cluster. Once we have the cluster running, we can start deploying cluster nodes, Pods (a collection of application containers), and containers. In terms of consistently managing Kubernetes platforms across multi-cloud platforms, there are multiple agnostic solutions that you can look at, such as Rancher or VMware’s Tanzu Mission Control.

For storage, we will have to manage, at a minimum, the following:

* Provisioning, configuring, and the operations of storage, including disks for managed virtual machines
* Detecting, investigating, and resolving incidents related to the functions of storage resources
* Monitoring, logging, health checks on local and different redundant types of storage solutions (zone, regional, globally redundant), and proactive checks/maintenance, including capacity checks and adjustments (capacity allocation)

Next, we will have to manage the accounts and make sure that our landing zone—the cloud environment and all its building blocks—is secure. Account management involves creating accounts or account groups that need access to the cloud environment. These are typically created in Active Directory.

In the final section of this chapter,*Understanding the need for demarcation*, we will take a deeper look at admin accounts and the use of global admin accounts. Security is tightly connected to account, identity, and access management, but also to things such as hardening (protecting systems from outside threats), endpoint protection, and vulnerability management. From day 1, we must have security in place on all the layers in order to prevent, detect, assess, and mitigate any breaches. This is part of SecOps. *Section 4* of this book is all about securing our cloud environments.

**Managing the base infrastructure**

After we have defined and deployed a landing zone in a public cloud platform, there are quite a number of building blocks that we will have to manage from that point onward. In this section, we will discuss how we can manage these building blocks. Firstly, we need to be able to connect to the landing zone, and that requires connectivity.

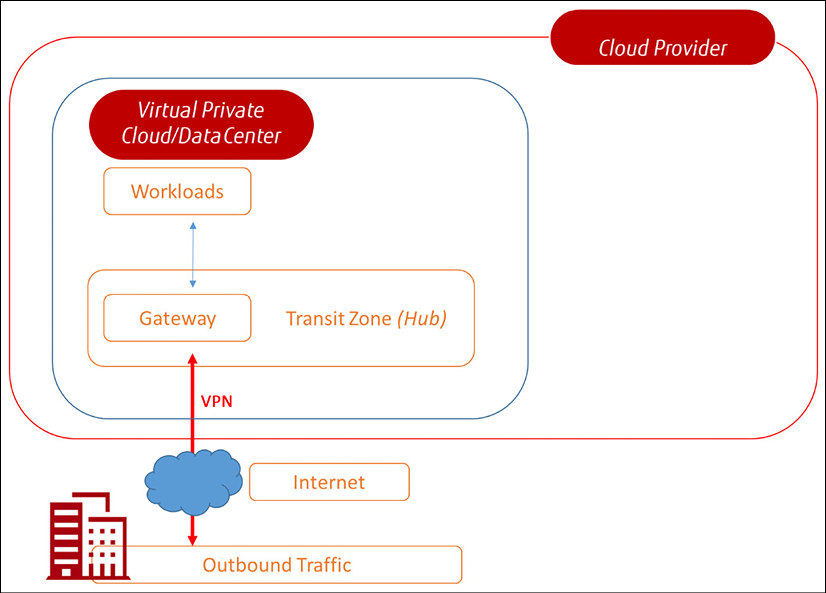
**Implementing and managing connectivity**

One of the most used technologies in cloud technology is the **VPN**, the **virtual private network**. In essence, a VPN is a tunnel using the internet as a carrier. It connects from a certain IP address or IP range to the IP address of a gateway server in the public cloud.

Before we get into this, you have to be aware of what a public cloud is. If you as a business deploy services in Azure, AWS, GCP, OCI, Alibaba Cloud, or any other public cloud, you are extending your data center to that cloud. It therefore needs a connection between your data center and that extension in the public cloud. The easiest and probably also the most cost-efficient way to get that connection fast is through a VPN. The internet is already there, and all you would have to do in theory is assign IP addresses or the IP range that is allowed to communicate to that extension, creating a tunnel. That tunnel can be between an office location (site) or from just one user connecting to the cloud. The latter is something we refer to as a point-to-site VPN.

In the public cloud itself, that connection needs to terminate somewhere, unless you want all resources to be opened up for connectivity from the outside. That is rarely the case and it’s certainly not advised. Typically, a business would want to protect workloads from direct and uncontrolled outside connections. When we’re setting up VPNs, we need to configure a zone in the public cloud with a gateway where the VPN terminates.

From the gateway, the traffic can be routed to other resources in the cloud, using routing rules and tables in the cloud. It works the same way as in a traditional data center where we would have a specific connectivity zone or even a **Demilitarized Zone** (**DMZ**) as a zone in the cloud environment that users connect to before they get to the actual systems. The following architecture shows the basic principle of a VPN connection to a public cloud:



*Figure 6.5: The basic architecture of VPN connectivity*

There are other ways to connect to a public cloud. VPN tunnels use the internet as a carrier. For several reasons, companies are often not very keen to have their traffic sent over the internet, not even when it’s through secured, encrypted tunnels. A more stable, predictable, and even more secure solution is a direct connection between the router or firewall in your on-premises environment and the cloud platform that you as a business use. Down to the core, this solution involves a cable from an on-premises network device straight into a gateway device that routes directly to services that your business purchases from the cloud. The leading cloud providers all offer direct connectivity solutions. Partner interconnects to the different clouds are also available through colocations such as Equinix, Digital Realty, and Interxion.

**Implementing Azure ExpressRoute**

Azure offers ExpressRoute as a direct connection from your network to your environments in Azure. ExpressRoute provides a private connection to Microsoft cloud services: Azure, Microsoft 365, and Dynamics. The offering comprises four different deployment types: cloud exchange co-location, point-to-point Ethernet, any-to-any IPVPN, and ExpressRoute Direct, which provides dual 100 Gbps or 10 Gbps connections supporting active/active connectivity:

* **Point-to-point Ethernet**: This type of connection provides connectivity on layer 2 or managed on layer 3. In the **Open Systems Interconnection** (**OSI**) model, layer 2 is the data link layer and layer 3 is the network layer. The main difference is that layer 3 provides routing and also takes care of IP routing, both static and dynamic, whereas layer 2 only does switching. Layer 3 is usually implemented when intra-**VLAN** (**Virtual Local Area Network**) is involved. Simply explained, layer 3 understands IP addresses and can route traffic on an IP base, whereas layer 2 does not understand IP addresses. Having said that, a point-to-point Ethernet ExpressRoute connection will typically—or is at least recommended to—be on layer 3.
* **Any-to-any IPVPN**: With this deployment, ExpressRoute integrates the **Wide-Area Network** (**WAN**) of the company with Azure, extending the on-premises environment with a virtual data center in Azure—making it literally one environment with the virtual data center as a *branch office*. Most companies will have network connectivity over **MPLS** (**Multiprotocol Label Switching**), provided by a telecom provider. ExpressRoute connects this network over layer 3 to Azure.
* **Azure cloud exchange co-location**: This will be the preferred solution for enterprises that host their systems in a co-location. If that co-location has a cloud exchange, you can use the connections from this exchange. Typically, these will be managed layer 3 connections. The hosted environment is connected to the exchange—often in so-called “meet-me rooms” or racks—and from the exchange, the connection is set up to Azure.

ExpressRoute lets customers connect directly to Microsoft’s network through a pair of 10 or 100 Gbps ports. Bandwidths are offered from 50 Mbps up to 10 Gbps.

**Implementing AWS Direct Connect**

The direct connection from your on-premises network to AWS is very appropriately called AWS Direct Connect. It connects your router or firewall to the Direct Connect service in your AWS Region. The connection goes from your own router to a router of your connectivity partner and from there, it gets directly linked to the Direct Connect endpoint.

That endpoint connects to a **Virtual Private Gateway**(**VPG**) in your AWS **Virtual Private Cloud**(**VPC**) or to the AWS services.

**Implementing Google Dedicated Interconnect**

Google offers Dedicated Interconnect. The principle is the same as with Direct Connect from AWS: from the router in the on-premises network, a direct link is established to the Google peering edge; this is done in a co-location facility where Google offers these peering zones. From there, the connection is forwarded to a cloud router in the GCP environment.

Dedicated Interconnect is offered as a single 10 Gbps or 100 Gbps link, or a link bundle that connects to the cloud router. Multiple connections from different locations or different devices to Google require separate interconnects.

**Implementing Alibaba Express Connect**

Alibaba Cloud offers Express Connect. Like the solutions that we discussed already, Express Connect also provides a dedicated, physical connection from a customer’s data center to Alibaba Cloud. The service is available worldwide.

**Implementing direct connectivity in OCI**

In OCI, there are several options available for direct connectivity to OCI resources. These options are designed to provide high-bandwidth, low-latency connectivity between on-premises data centers, colocation facilities, or other cloud providers.

FastConnect is a dedicated, private connectivity option that provides high-bandwidth and low-latency connections between on-premises data centers or colocation facilities and OCI. FastConnect provides dedicated, private connections up to 100 Gbps and allows you to bypass the public internet.

An alternative is VPN Connect: This option provides an encrypted and secure connection between an on-premises network and a VCN in OCI. VPN Connect uses the internet to establish a secure connection, making it a more cost-effective option compared to FastConnect. VPN Connect is available in different configurations, such as site-to-site VPN and remote access VPN, to suit different connectivity needs.

Lastly, we mention Oracle Interconnect as an option to set up private connectivity between OCI and other cloud providers, such as Azure and AWS. Oracle Interconnect provides dedicated, private connections up to 10 Gbps.

**Accessing environments in public clouds**

Once we have connectivity, we need to design how we access the resources in the cloud. We don’t want admins to access every single resource separately but through a single point of access so that we can control access on that single instance. Steppingstone or bastion servers are a solution for this.

Steppingstone servers are sometimes referred to as jump servers. Azure, AWS, and GCP call these servers bastion servers. The idea is to allow access to one or more network segments through a select number of machines, in this case, the jump server(s). This not only limits the number of machines that can access those network segments, but also the type of traffic that can be sent to those network segments. It’s highly recommended and best practice to deploy these jump or steppingstone servers in a virtual network in the cloud.

**Defining and managing infrastructure automation tools and processes**

In the cloud, we work with code. There’s no need to buy physical hardware anymore; we simply define our hardware in code. This doesn’t mean we don’t have to manage it. To do this in the most efficient way, we need a master code repository. This repository will hold the code that defines the infrastructure components, as well as how these components have to be configured to meet our principles in terms of security and compliance. This is what we typically refer to as the desired state.

Azure, AWS, and Google offer native tools to facilitate infrastructure and configuration as code, as well as tools to automate the deployment of the desired state. In Azure, we can work with Azure DevOps, GitHub Actions, and Azure Automation, all of which work with **Azure Resource Manager** (**ARM**). AWS offers CloudFormation, while Google has Cloud Resource Manager and Cloud Deployment Manager. These are all tied into the respective platforms, but the market also offers third-party tooling that tends to be agnostic to these platforms. We will explore some of the leading tools later in this chapter.

For source code management, we can use tools such as GitHub, Azure DevOps, AWS CodeCommit, and GCP Cloud Source Repositories.

**Defining and implementing monitoring and management tools**

We’ve already discussed the need for monitoring. The next step is to define what tooling we can use to perform these tasks. Again, the cloud platforms offer native tooling: Azure Monitor, Application Insights, and Log Analytics; AWS CloudTrail and CloudWatch; and Google Cloud’s operations suite (formerly Stackdriver). And, of course, there’s a massive set of third-party tools available, such as Splunk and Nagios. These latter tools have a great advantage since they can operate independently of the underlying platform. This book won’t try to convince you that tool A is preferred over tool B; as an architect, you will have to decide what tool fits the requirements—and the budget, for that matter.

Security is a special topic. The cloud platforms have spent quite some effort in creating extensive security monitoring for their platforms. Monitoring is not only about detecting; it’s also about triggering mitigating actions. This is especially true when it comes to security where detecting a breach is certainly not enough. Actually, the time between detecting a vulnerability or a breach and the exploit can be a matter of seconds, which makes it necessary to enable fast action. This is where **SIEM** comes into play: **security incident and event management**. SIEM systems evolve rapidly and, at the time of writing, intelligent solutions are often part of the system.

An example of this is Azure Sentinel, an Azure-native SIEM and **SOAR** (**Security Orchestration, Automation and Response**) solution: it works together with Azure Security Center, where policies are stored and managed, but it also performs an analysis of the behavior it sees within the environments that an enterprise hosts on the Azure platform. Lastly, it can automatically trigger actions to mitigate security incidents. For instance, it can block an account that logs in from the UK one minute and from Singapore the next—something that wouldn’t be possible without warp-driven time traveling. As soon as it monitors this type of behavior, an action might be triggered to block the user from the system.

In other words, monitoring systems become more sophisticated and developments become as fast as lightning.

**Supporting operations**

Finally, once we have thought about all of this, we need to figure out who will be executing all these tasks. We will need people with the right skills to manage our multi-cloud environments. As we have said already, a truly T-shaped engineer or admin doesn’t exist. Most enterprises end up with a group of developers and operators that all have generic and more specific skills.

Some providers refer to this as the **Cloud Center of Excellence** (**CCoE**), and they mark it as an important step in the cloud journey or cloud adoption process of that enterprise. Part of this stage would be to identify the roles this CCoE should have and get the members of the CCoE on board with this. The team needs to be able to build and manage the environments, but they will also have a strong role to fulfill to evangelize new cloud-native solutions.

Just as a reading tip, please have a look at an excellent blog post on forming a CCoE by Amazon’s Enterprise Strategist Mark Schwartz: <https://aws.amazon.com/blogs/enterprise-strategy/using-a-cloud-center-of-excellence-ccoe-to-transform-the-entire-enterprise/>.

In this section, we have learned what we need to cover to set up our operations in multi-cloud. The next step is managing the landing zones.

**Managing the landing zones using policies**

When we work in cloud platforms, we work with code. Everything we do in the cloud is software-and code-defined. This makes cloud infrastructure very agile, but it also means that we need some strict guidelines for how we manage the code, starting with the code that defines our landing zone or foundation environment. As with everything in IT, it needs maintenance. In traditional data centers and systems, we have maintenance windows where we can update and upgrade systems. In the cloud, things work a little differently.

First, the cloud providers apply maintenance whenever it’s needed. There’s no way that they can agree upon maintenance windows with thousands of customers spread across the globe. They simply do whatever needs to be done to keep the platform healthy and ready for improvements and the release of new features. Enterprises don’t want to be impacted by these maintenance activities, so they will have to make sure that their code is always safe.

The next thing we need to take into account is the systems that the enterprise has deployed on the platform, within their own virtual cloud or project. These resources also need maintenance. If we’re running VMs, we will need to patch them every now and then. In this case, we are patching code. We want to make sure that, with these activities, administrators do not accidentally override certain security settings or, worse, delete disks or any critical code that is required for a specific function that a resource fulfills. This is something that we must think about from the very start when setting up the landing zones. From that point onward, we must start managing them. For that, we can use policies and management tooling.

In this section, we have set up the landing zones. In the next section, we’ll learn how to manage them.

**Managing basic operations in Azure**

When we look at Azure, we must look at a service called **Test-Driven Development** (**TDD**) for landing zones in Azure. Be aware that TDD is not an Azure service in itself: it’s an agile practice used in software development that can be applied to the development of landing zones in the cloud. TDD is particularly known in software development as it aims to improve the quality of software code. As we have already discussed, the landing zone in Azure is expanded through the process of refactoring, an iterative way to build out the landing zone. Azure provides a number of tools that support TDD and help in the process of refactoring the landing zone:

* **Azure Policy**: This validates the resources that will be deployed in Azure against the business rules. Business rules can be defined as cost parameters or thresholds, as well as security parameters such as checking for the hardening of resources or consistency with other resources. For instance, they can check if a certain ARM template has been used for deployment. Policies can also be grouped together to form an initiative that can be assigned to a specific scope, such as the landing zone. A policy can contain actions, such as denying changes to resources or deploying after validation. Azure Policy offers built-in initiatives that can be specifically used to execute TDD: it will validate the planned resources in the landing zone against business rules. A successful validation will result in a so-called definition of done and, with that, acceptance that resources may be deployed.
* **Azure Blueprints**: With Blueprints, you can assemble role assignments, policy assignments, ARM templates, and resource groups in one package so that they can be reused over and over again in case an enterprise wants to deploy multiple landing zones in different subscriptions. Microsoft Azure offers various blueprint samples, including policies for testing and deployment templates. The good thing is that these can easily be imported through Azure DevOps so that you have a CI/CD pipeline with a consistent code repository right from the start.
* **Azure Graph**: Azure Landing Zone is deployed based on the principle of refactoring. So, in various iterations, we will be expanding our landing zone. Since we are working according to the principles of TDD, this means that we must test whether the iterations are successfully implemented, that resources have been deployed in the right manner, and that the environments have interoperability. For these tasks, Azure offers Graph. It creates test sets to validate the configuration of the landing zone. Azure Graph comes with query samples, since it might become cumbersome to get started with the settings and coding that Graph uses.
* **Azure Quickstart Templates**: If we really want to get going fast, we can use Quickstart Templates, which provide default settings for the deployment of the landing zone itself and its associated resources.
* **Azure Bicep**: This is a **Domain-Specific Language**(**DSL**) that uses declarative syntax to deploy Azure resources. It was introduced in 2020. In a Bicep file, you define the infrastructure you want to deploy to Azure and then use that file throughout the development lifecycle. The file can be used to deploy infrastructure repeatedly, but always in a consistent manner. The big advantage is that developers and administrators only have to manage the Bicep file that is used to deploy resources. Bicep files have an easier syntax than **JSON**(**JavaScript Object Notation**) code. The Bicep syntax directly communicates with **Azure Resource Manager**(**ARM**) so that resources are always implemented in the most optimized way. In a way, Bicep is comparable to Terraform, a tool that provides an automated way to deploy **Infrastructure as Code**(**IaC**) too.

More information on test-driven development in Azure Landing Zone can be found at <https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/ready/considerations/azure-test-driven-development>.

**Managing basic operations in AWS**

AWS offers CloudFormation guardrails. This is a very appropriate name since they really keep your environment on the rails. Guardrails come with four principal features for which policies are set in JSON format. To create policies, AWS offers Policy Generator. In Policy Generator, you define the type of policy first and then define the conditions, meaning when the policy should be applied. The following policies are in scope:

* **Termination protection**: Here, AWS talks about stacks and even nested stacks. Don’t get confused—a stack is simply a collection of AWS resources that can be managed as one unit from the AWS Management Console. An example of a stack can be an application that comprises a frontend server, a database instance using an S3 bucket, and network rules. Enabling termination protection prevents that stack from being deleted unintentionally. Termination protection is disabled by default, so you need to enable it, either from the management console or by using command-line programming.
* **Deletion policies**: Where termination protection has entire stacks as the scope, deletion policies target specific resources. To enable this, you must set DeletionPolicy attributes within the CloudFormation templates.

Now, this policy comes with a lot of features. For instance, the policy has a retain option so that whenever a resource is deleted, it’s still kept as an attribute in your AWS account. You can also have CloudFormation take a snapshot of the resource before it gets deleted. Keep in mind that deletion policies are set per resource.

* **Stack policies**: These policies are set to define actions for a whole stack or group of resources. Stack policies are a type of access control policy that aims to protect critical infrastructure resources by preventing updates to them. Stack policies are applied to AWS CloudFormation stacks, which are templates that define the infrastructure resources that make up an application. Stack policies work by specifying a set of rules that restrict the types of updates that can be made to specific resources within a stack. For example, we might use a stack policy to prevent updates to a production database or to restrict changes to your load balancer configuration.
* **IAM policies**: These policies define access controls; that is, who is allowed to do what and when? Access controls can be set with fine granularity for whole stacks, specific resource groups, or even single resources and only allow specific tasks to define the roles that users can have. In other words, this is the place where we manage **Role-Based Access Control** (**RBAC**). The last section of this chapter, *Understanding the need for demarcation*, is all about IAM and the separation of duties.

One important service that must be discussed here is AWS Control Tower. In most cases, enterprises will have multiple accounts in AWS, corresponding with divisions or teams in the enterprise. On an enterprise level, we want to keep the AWS setup consistent, meaning that guardrails are applied in the same way in every account. Control Tower takes care of that.

Control Tower is a service that creates a landing zone with AWS Organizations and helps in setting up multi-account governance, based on the AWS best practices. These best practices obviously comply with the CAF and the WAF. So, with Control Tower you set up a landing zone based on best-practice blueprints from AWS Organizations and apply the guardrails to this landing zone. Blueprints contain policies for identity management, federated access to accounts, centralized logging, automated workflows for the provisioning of accounts, and network configurations.

Next, the administrator gets a dashboard in Control Tower that shows how accounts and guardrails are applied.

More information on guardrail policies in AWS can be found at <https://aws.amazon.com/blogs/mt/aws-cloudformation-guardrails-protecting-your-stacks-and-ensuring-safer-updates/>. Information about Control Tower is provided at <https://aws.amazon.com/controltower/?control-blogs.sort-by=item.additionalFields.createdDate&control-blogs.sort-order=desc>.

**Managing basic operations in GCP**

As we have seen, GCP can be a bit different in terms of the public cloud and landing zones. This originates from the conceptual view that Google has, which is more focused on container technology using Kubernetes. Still, GCP offers extensive possibilities in terms of setting policies for environments that are deployed on GCP. In most cases, these policies are comprised of organizations and resources that use IAM policies:

* **Organizations**: In GCP, we set policies using constraints. A constraint is an attribute that is added to the service definition. Just as an example, we’ll take the Compute Engine service that deploys VMs to our GCP project. In Compute Engine projects, logging in for operating systems is disabled by default. We can enable this and set a so-called Boolean constraint, named after George Boole, who invented this type of logic as an algebraic system in the nineteenth century: a statement or logical expression is either true or false. In this case, we set Compute Engine to true. Next, we must set a policy that prevents that login from being disabled. The command in gcloud + code style is: constraints/compute.requireOsLogin. A lot of policies and constraints in GCP work according to this principle.

More on organization policy constraints in GCP can be found at <https://cloud.google.com/resource-manager/docs/organization-policy/org-policy-constraints>.

* **Resource policies**: Cloud IAM policies set access controls for all GCP resources in JSON or YAML format. Every policy is defined by bindings, an audit configuration, and metadata. This may sound complex, but once you understand this concept, it does make sense. First, let’s look at bindings. Each binding consists of a member, a role, and a condition. The member can be any identity. Remember what we said previously: in the cloud, basically, everything is an identity. This can be users, but also resources in our cloud environment that have specific tasks so that they can access other resources and have permission to execute these tasks.

Thus, a member is an identity: a user, a service account, a resource, or a group of resources. The member is bound to a role that defines the permission that a member has. Finally, we must determine under what condition a member may execute its role and what constraints are valid. Together, this makes a binding.

However, the binding is only one part of the policy. We also have an AuditConfig to log the policy and the metadata. The most important field in the metadata is etag. The etag field is used to guarantee that policies are used in a consistent way across the various resources in the project. If a policy is altered on one system, the etag field makes sure that the policies stay consistent. Inconsistent policies will lead resource deployments to fail.

Policies can have multiple bindings and can be set on different levels within GCP. However, be aware that there are limitations. As an example, GCP allows a maximum of 1,500 members per policy. So, do check the documentation thoroughly, including the best practices for using policies.

Extensive documentation on Cloud IAM policies in GCP can be found at <https://cloud.google.com/iam/docs/policies>.

**Managing basic operations in Alibaba Cloud**

Managing resources in Alibaba Cloud is done best through the Cloud Governance Center service. Alibaba recommends setting this up with a corporate account or the RAM user: the Resource Access Management user. Note that is not RAM as in random access memory. RAM in Alibaba Cloud is related to resource access.

The RAM user gets permission to access the various resources in Alibaba Cloud. Next, the resource structure is defined by setting up the resource directory and the management accounts. The resource directory holds the resource folders with, for instance, the **Elastic Compute Service** (**ECS**) instances and the storage accounts. Also, shared service and logging accounts are defined.

Administrators are advised to use Account Factory, where the baselines—the guardrails—for identities, permissions, networking, and security are set. Cloud Governance Center offers a way to centrally manage these items and make sure that resources are configured in a consistent manner.

This completes this section, in which we have learned how to create policies by enabling the **Basic Operations** (**BaseOps**) of our landing zones in the different clouds. The next section talks about orchestrating policies in a true multi-cloud setup, using a single repository.

**Managing basic operations in OCI**

To start managing resources in **Oracle Cloud Infrastructure** (**OCI**), we need to create an OCI account and obtain the necessary credentials to access the OCI console and APIs.

First, we need an OCI account to access the OCI console and APIs. This account provides access to the OCI services and resources that we need to manage our infrastructure in OCI. Next, we create user accounts for each person who will be managing resources in OCI. User accounts are associated with a specific OCI identity domain and can be granted specific permissions to access and manage resources within that domain.

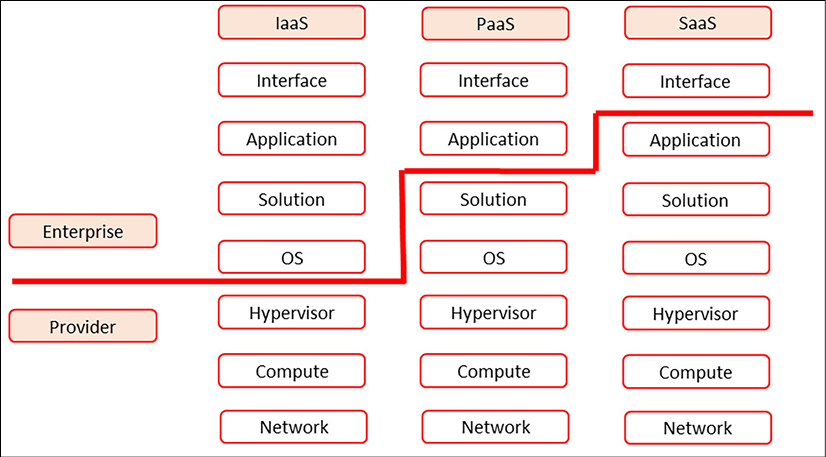
We also need to define groups. We can create groups within the OCI identity domain to organize users based on their roles and responsibilities. Groups can be used to simplify permissions management and make it easier to grant or revoke access to resources.

Resources are organized in compartments: logical containers for resources in OCI. They provide a way to organize and isolate resources based on their business purpose or application. We must create compartments within the OCI tenancy and assign permissions to users and groups to access and manage resources within those compartments.

Finally, we must set up an API signing key. To access the OCI APIs, we need to generate an API signing key for each user account. This key is used to authenticate API requests and authorize access to specific resources.

**Understanding the need for demarcation**

Typically, when we talk about demarcation in cloud models, we refer to the matrix or delineation of responsibility: who’s responsible for what in IaaS, PaaS, and SaaS computing? The following diagram shows the very basics of this matrix—the shared responsibility model:



*Figure 6.6: Shared responsibility model in cloud deployment*

However, we need a much more granular model in multi-cloud. We have been discussing policies throughout this chapter and by now, we should have concluded that it’s not very easy to draw some sharp lines when it comes to responsibilities in our multi-cloud environment. Just look at the solution stack: even in SaaS solutions, there might be certain security and/or compliance policies that the solution needs to adhere to. Even something such as an operating system might already be causing issues in terms of hardening: are monitoring agents from a PaaS provider allowed or not? Can we run them alongside our preferred monitoring solution? Or will that cause too much overhead on our systems? In short, the world of multi-cloud is not black and white. On the contrary, multi-cloud has an extensive color scheme to work with.

So, how do we get to a demarcation model that will work for our enterprise? Well, that’s architecture. First, we don’t need global admins all over our estate. This is a major pitfall in multi-cloud. We all know such cases: the database administrator that needs global admin rights to be able to execute certain actions or, worse, solutions that require service accounts with such roles. It’s global admin galore. Do challenge these requests and do challenge software providers—or developers, for that matter—when it comes to why systems would need the highest possible access rights in the environment.

That’s where it starts: policies. In this case, a good practice is the **Policy of Least Privilege** (**PoLP**). This states that every identity is granted the minimum amount of access that is necessary to perform the tasks that have been assigned to that identity. Keep in mind that an identity, in this case, doesn’t have to be a user: it can be any resource in the environment. When we are talking about users, we refer to this as **Least-Privileged User Account or Access** (**LPUA**). PoLP helps in protecting data as data will only be accessible when a user or identity is explicitly granted access to that data. But there are more reasons to adhere to this policy. It also helps in keeping systems healthy as it minimizes risks or faults in systems. These faults can be unintended or the result of malicious conduct. We should follow the rule of least privilege at all times. We will discuss this in more detail in *Chapter 15*, *Implementing Identity and Access Management*, which is all about identity and access management.

Regarding this very first principle, there are a few more considerations that need to be made at this stage. These considerations translate into controls and, with that, into deliverables that are part of BaseOps, since they are absolutely part of the foundational principles in multi-cloud. The following table shows these controls and deliverables:

|  |  |
| --- | --- |
| **Control** | **Deliverable** |
| A policy document is available and active that describes how user/admin accounts are generated, maintained, and disposed of throughout their lifecycle. | Policy and approvals |
| An RBAC authorization matrix is available that describes the access delegation from data or system owners. | Authorization matrix |
| User accounts are created by following established approval procedures that adhere to LPUA. | List of user accounts |
| Periodic checks are performed to ensure the continuous validity of accounts; for example, the account is needed and in active use, the correct role has been applied, and so on. | Checklist |
| Unauthorized access (attempts) to system resources is logged in an audit trail and periodically reported to and reviewed by the CISO/Security Manager. | Report of unauthorized access attempts |

*Table 6.1 – Cloud controls and associated deliverables*

Demarcation and separation of duties are very strongly related to identity and access management. We will learn more about that in the forthcoming sections about security.

**Summary**

In this chapter, we have designed and set up our landing zones on the different major cloud platforms. We have learned that the foundational principles might be comparable, but the actual underlying implementation of the landing zone concepts differs. We studied these various concepts, such as enterprise-scale in Azure and Control Tower in AWS.

We also learned how to work with patterns to define the components of a landing zone, starting with connectivity. VPN connections are still the most commonly used way to connect to public clouds, but enterprises might also choose direct dedicated connections. In this chapter, the various direct connectivity offerings of Azure, AWS, GCP, Alibaba Cloud, and OCI have been discussed. As soon as we have connectivity, we can start managing workloads in the cloud. The main concern for administrators is to keep resources and configurations consistent in different deployments.

Finally, we learned that there’s a need for a redundant demarcation model in multi-cloud. This all adds up to the concept of BaseOps: getting the basics right.

Part of getting the basics right is making sure that our environments are resilient and performing well. That’s what we will be discussing in the next chapter, where we will concentrate on risks and how to mitigate these.

**Questions**

1. A basic infrastructure in the cloud consists of five major domains, three of which are network, compute, and storage. What are the other two domains?
2. What solution does Azure offer to provide scaling of landing zones?
3. AWS offers a service that enables the central management of guardrails for resources. What’s the name of this service?
4. What does RAM stand for in Alibaba Cloud?

**Further reading**

* *Azure for Architects*, by Ritesh Modi, Packt Publishing
* *Architecting C*