**7**

**Designing Applications for Multi-Cloud**

An important topic in any multi-cloud architecture is the resilience and performance of the applications that we will host in the cloud. We will have to decide on the type of solution that fits the business requirements and mitigates the risks of environments not being available, not usable, or not secured. Solutions and concepts such as 12-factor apps, PaaS, and SaaS must be considered as strategies.

In this chapter, we will gather and validate business requirements for resilience and performance. We will learn how to use tools to build resiliency in cloud platforms and how to optimize the performance of our application environments using advisory tools and support plans.

We will cover the following topics in this chapter:

* Architecting for resilience and performance
* Starting with business requirements
* Using the principles of the 12-factor app
* Accelerating application design with PaaS
* Designing SaaS solutions
* Performance KPIs in a public cloud—what’s in it for you?
* Optimizing your multi-cloud environment
* Use case: creating solutions for business continuity and disaster recovery

**Architecting for resilience and performance**

Resilience and performance are two important factors in application architecture. Cloud platforms offer great tools to enhance both resilience and performance. Typically, these factors are included in the Well-Architected Frameworks that we discussed in the previous chapter. But what do we mean by resilience and performance?

We can use the definition that AWS uses for resilience in its Well-Architected Framework: *the capability to recover when stressed by load, attacks, and failure of any component in the workload’s components*. To simplify this a bit: if any component of a system fails, how fast and to what extent can it recover, preferably to its original state, without loss of data. Obviously, an organization will need to test that, and that’s where Chaos Monkey comes in as a testing strategy. In *Chapter 18*, *Developing for Multi-Cloud with DevOps and DevSecOps*, we will discuss testing strategies in more detail since they are part of the DevSecOps Maturity Model.

Defining the resilience of systems starts with understanding the criticality to the business of that system and what service levels need to be applied. That begins with having a clear understanding of business requirements. But it’s important to realize one major thing: resilience and performance are not strictly technical issues. It’s also about operational responsibility, which is a shared responsibility across all teams that build and manage applications.

**Starting with business requirements**

In *Chapter 6*, *Controlling the Foundation Using Well-Architected Frameworks*, we talked a little bit about things such as availability, backup, and disaster recovery. In this chapter, we will take a closer look at the requirements and various solutions that cloud platforms offer to make sure that your applications are available, accessible, and, most of all, safe to use. Before we dive into these solutions and the various technologies, we will have to understand what the potential risks are for our business if we don’t have our requirements clearly defined.

In multi-cloud, we recognize risks at various levels, again aligning with the principles of enterprise architecture.

**Understanding data risks**

The biggest risk concerning data is ambiguity about the ownership of the data. This ownership needs to be regulated and documented in contracts, as well as with the cloud providers. International and national laws and frameworks such as the **General Data Protection Regulation** (**GDPR**) already define regulations in terms of data ownership in Europe, but nonetheless, be sure that it’s captured in the service agreements as well. First, involve your legal department or counselor in this process.

We should also make a distinction between types of data. Is it business data, metadata, or data that concerns the operations of our cloud environments? In the latter category, you can think of the monitoring logs of the VMs that we host in our cloud. For all these kinds of data, there might be separate rules that we need to adhere to in order to be compliant with laws and regulations.

We need to know and document where exactly our data is. Azure, AWS, and GCP have global coverage and will optimize their capacity as much as they can by providing resources and storage from the data centers where they have that capacity. This can be a risk. For example, a lot of European countries specify that specific data cannot leave the boundaries of the European Union (EU). In that case, we will need to ensure that we store data in cloud data centers that are in the EU. So, we need to specify the locations that we use in the public cloud: the region and the actual country where the data centers reside.

Recently, the US implemented a new law protecting European citizens from unauthorized usage of data. The possibility that privacy-sensitive data from European citizens could be tapped by American companies—including the major cloud providers Microsoft, AWS, and GCP, which provide a lot of services to European companies and individuals—was an immense legal issue. In September of 2022, US President Joe Biden signed a bill stating that an independent institute—not related to the US government—has to judge whether European data is well-enough protected from unauthorized usage. The debate about this shows how complicated this matter is when data is collected, used, and shared across global cloud platforms.

We also need to ensure that when data needs to be recovered, it’s recovered in the desired format and in a readable state. Damaged or incomplete data is the risk here. We should execute recovery tests on a regular basis and have the recovery plans and the actual test results audited. This is to ensure that the data integrity is guarded at all times. This is particularly important with transaction data. If we recover transactions, we need to ensure that all the transactions are recovered, but also that the transactions are not replicated during the recovery procedure. For this, we also need to define who’s responsible for the quality of the data, especially in the case of SaaS.

To help with structuring all these requirements, a good starting point would be to create a model for data classification. Classification helps you decide what type of solution needs to be deployed to guarantee the confidentiality, integrity, and availability of specific datasets. Some of the most commonly used data categories are public data, confidential company data, and personal data.

**Understanding application risks**

The use of SaaS is becoming increasingly popular. Many companies have a strategy that prefers SaaS before PaaS before IaaS. In terms of operability, this might be the preferred route to go, but SaaS does come with a risk. In SaaS, the whole solution stack is managed by the provider, including the application itself. A lot of these solutions work with shared components, and you have to combat the risk of whether your application is accessible to others or whether your data can be accessed through these applications. A solution to mitigate this risk is to have your own application runtime in SaaS.

One more risk that is associated with the fact that the whole stack—up until the application itself—is managed by the provider is that the provider can be forced out of business. At the time of writing, the world is facing the coronavirus pandemic, and a lot of small businesses are really struggling to survive. We are seeing businesses going down, and even in IT, it’s not always guaranteed that a company will keep its head above water when a severe crisis hits the world. Be prepared to have your data safeguarded whenever a SaaS provider’s propositions might be forced to stop the development of the solution or, worse, to close down the business.

Also, there’s the risk that the applications fail and must be restored. We have to make sure that the application code can be retrieved and that applications can be restored to a healthy state.

**Understanding technological risks**

We are configuring our environments in cloud platforms that share a lot of components, such as data centers, the storage layer, the compute layer, and the network layer. By configuring our environment, we merely **silo**. This means that we are creating a separate environment—a certain area on these shared services. This area will be our virtual data center. However, we are still using the base infrastructure of Azure, AWS, GCP, or any other cloud.

Even the major cloud providers can be hit by outages. It’s up to the enterprise to guarantee that their environments will stay available, for instance, by implementing redundancy solutions when using multiple data centers, zones, or even different global regions.

Observability and monitoring are a must, but they don’t have any added value if we’re looking at only one thing in our stack or at the wrong things. Bad monitoring configuration is a risk. As with security, the platforms provide their customers with tools, but the configuration is up to the company that hosts its environment on that platform.

Speaking of security, one of the biggest risks is probably weak security. Public clouds are well protected as platforms, but the protection of your environment always remains your responsibility. Remember that clouds are a wonderful target for hackers since they’re a platform hosting millions of systems. That’s exactly the reason why Microsoft, Amazon, and Google spend a fortune securing their platforms. Make sure that your environments on these platforms are also properly secured and implement endpoint protection, hardening of systems, network segmentation, firewalls, and vulnerability scanning, as well as alerting, intrusion detection, and prevention. You also need to ensure you have a view of whatever is happening on the systems.

However, do not overcomplicate things. Protect what needs to be protected but keep it manageable and comprehensible. The big question is, what do we need to protect, to what extent, and against which costs? This is where gathering business requirements begins.

Although we’re talking about technological, application, and data risks, at the end of the day, it’s about business requirements. These business requirements drive data, applications, and technology, including risks. So far, we haven’t answered the question of how to gather these business requirements.

The main goal of this process is to collect all the relevant information that will help us create the architecture, design our environments, implement the right policies, and configure our multi-cloud estate as the final product. Now, this is not a one-time exercise. Requirements will change over time, and especially in the cloud era, the demands and, therefore, the requirements are constantly changing at an ever-increasing speed. So, gathering requirements is an ongoing and repetitive process.

How do we collect the information we need? There are a few key techniques that we can use for this:

* **Assessment**: A good way to do this is to assess whether we’re assessing resilience and performance from the application layer. What does an application use as resources, and against which parameters? How are backups scheduled and operated? What are the restore procedures? Is the environment audited regularly, what were the audit findings, and have these been recorded, scored, and solved? We should also include the end-user experience regarding the performance of the application and under what conditions, such as office rush hour, when the business day starts, and normal hours.
* **Stakeholder interviews**: These interviews are a good way to understand what the business need is about. We should be cautious, though. Stakeholders can have different views on things such as what the business-critical systems are.
* **Workshops**: These can be very effective for drilling down a bit deeper into the existing architectures, the design of systems, the rationales behind demands, and the requirements, while also giving us the opportunity to enforce decisions since all stakeholders will ideally be in one room. A risk of this is that discussions in workshops might become too detailed. A facilitator can help steer this process and get the desired result.

Once we have our requirements, then we can map to the functional parameters of our solution. A business-critical environment can have the requirements that it needs to be available 24/7, 365 days a year. The system may hold transactions where every transaction is worth a certain amount of money. Every transaction that’s lost means that the company is losing money. The systems handle a lot of transactions every minute, so every minute of data loss equals an amount of real financial damage. This could define the **recovery point objective** (**RPO**)—the maximum amount of data loss the company finds acceptable—which should be close to 0. This means that we have to design a solution that is highly available, redundant, and guarded by a disaster recovery solution with a restore solution that guarantees an RPO of near 0—possibly a solution that covers **data loss prevention** (**DLP**).

Is it always about critical systems, then? Not necessarily. If we have development systems wherein a lot of developers are coding, the failure of these systems could actually trigger a financial catastrophe for a company. The project gets delayed, endangering the time to market of new services or products, and the developers will sit idle, but the company will still have to pay them. It’s always about the business case, the risks a company is willing to accept, and the cost that the company is willing to pay to mitigate these risks.

**Using the principles of the 12-factor app**

The 12-factor app is a methodology for building modern apps, ready for deployment on cloud platforms, abstracted from server infrastructure. The major benefit of applying the 12-factors is that developers can create apps that are not only scalable, but also reliable since the app is self-contained. Dependencies are clearly defined and isolated from the app. This includes the dependencies on the underlying infrastructure.

Next, it allows for a consistent way of managing code through a single code base and a defined, declarative way of deployment.

The format of the 12-factor app is based on the book *Patterns of Enterprise Application Architecture and Refactoring* by Martin Fowler, published by Addison-Wesley Professional, 2005. Although it was written in 2005—a long time ago in terms of cloud technology—it is still very relevant in terms of its architecture. In cloud-native development, the 12-factor principles are widely used.

The twelve factors are as follows:

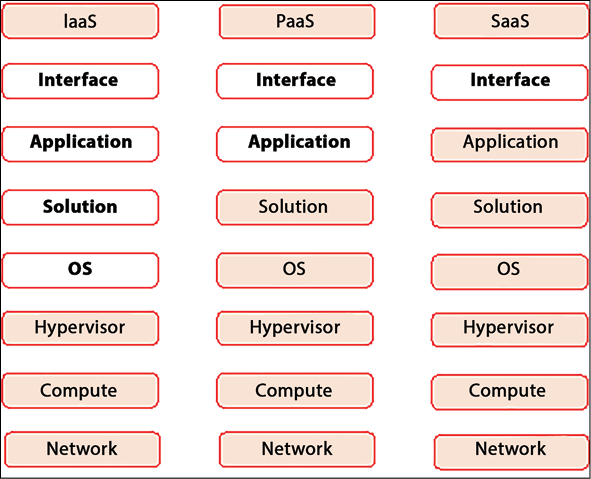
1. **Code base**: One code base is tracked in revision control, with many deployments (including Infrastructure as Code).
2. **Dependencies**: Explicitly declare and isolate dependencies.
3. **Config**: Store config in the environment (Configuration as Code).
4. **Backing services**: Treat backing services as attached resources. Think of databases and queues.
5. **Build, release, run**: Strictly separate build and run stages (pipeline management, release trains).
6. **Processes**: Execute the app as one or more stateless processes.
7. **Port binding**: Export services via port binding.
8. **Concurrency**: Scale out via the process model.
9. **Disposability**: Maximize robustness with fast startup and graceful shutdown.
10. **Dev/prod parity**: Keep development, staging, and production as similar as possible.
11. **Logs**: Treat logs as event streams.
12. **Admin processes**: Run admin/management tasks as one-off processes.

More on these principles can be found at [https://12factor.net](https://12factor.net/). We will find out that a number of these twelve principles are also captured in different frameworks and embedded in the governance of multi-cloud environments, such as DevOps and **Site Reliability Engineering** (**SRE**).

**Accelerating application design with PaaS**

**PaaS** is the abbreviation for **Platform as a Service**. The first question that we must answer then is: what is a platform? Opinions and definitions differ in this case. The best way to define a platform would be a service or collection of services that enable teams and organizations to speed up the development and delivery of software. To put it very simply: PaaS makes life easy for software teams.

With the use of PaaS, the teams do not need to bother about a wide variety of services that they need to build before they can start with the software product that really adds business value. In essence, PaaS is an abstraction layer on top of the infrastructure. Let’s have another look at the diagram that we discussed in the previous chapter, where we talked about demarcation:



*Figure 7.1: The differences in responsibilities between IaaS, PaaS, and SaaS*

In PaaS everything up until the operating system is taken care of by the platform. Developers only have to write code for the applications.

The container hosting platform Kubernetes fits the definition. Kubernetes manages the infrastructure for containers that are used to host applications. Developers only need to worry about the application code. The code is packaged into containers that land on the Kubernetes clusters. Kubernetes, as a platform, makes sure that the underlying infrastructure is available. The major advantages of using PaaS are:

* **Limit complexity**: PaaS is on a higher abstraction layer than IaaS.
* **Efficient onboarding for applications**: A few bloggers will call this the “paved way for shipping stuff.” There’s no need to configure infrastructure: we only need to build the application; the platform does the rest.
* **Consistency**: Since everything is taken care of by the platform, the underlying infrastructure remains consistent across the various development teams.
* **Compliancy**: Compliancy is built into the platform services. Guardrails are implemented and managed at the platform level.

The use of PaaS can accelerate the development of applications, but there’s a downside. Platforms require heavy standardization, and this may lead to restrictions for the developers of applications.

**Designing SaaS solutions**

The market for **Software as a Service (SaaS)** has been growing with massive numbers over the past five years. It’s expected that the market for **SaaS** will grow to over 600 billion US dollars during the year 2023.

SaaS as a strategy makes sense if a company wants to expand its business fast. SaaS products allow customers access to applications over the internet without needing to buy a physical product that they need to install themselves on a device. SaaS products are centrally hosted on cloud platforms, where the product is fully managed by the developers. The customer is assured of the newest releases because of the central hosting and management model.

SaaS is extremely customer-centric: the customer simply subscribes to the service, gets access, and, next, the service is ready to use on the device of the customer’s choice. Hence, developing SaaS products has become sort of the holy grail for many enterprises.

The major cloud providers offer toolkits to develop SaaS functionality on their platforms. AWS calls this the SaaS Factory, and Azure has the SaaS Development Kit. All these services aim to help companies transform monolithic applications into modern web-based applications that can be delivered as SaaS. At re:Invent 2020, the main convention of AWS, SaaS Boost was released. In 2022 version 2 was released. SaaS Boost provides out-of-the-box and ready-to-use core components to build software. It uses CloudFormation to dynamically create and deploy resources to run application code, using container infrastructure with **Elastic Container Services (ECS)** and Lambda serverless functions to orchestrate workflows between the various components.

The magic word in SaaS is *easy*. Subscribing and onboarding to SaaS should be easy and extremely user-friendly. The user interface, typically using a web interface, must be accessible and easy to use. Most important: SaaS must be available. Think of Netflix and Spotify: these services are always “on.” They can’t allow for major disruptions in the services. These environments are built as micro-services, allowing parts of the services to be continuously updated and upgraded in CI/CD pipelines, without causing downtime for the customers. Keep in mind that customers pay for these services in the form of subscription fees. Major outages will seriously cause trust issues with customers.

SaaS applications must be designed with the customer and the customer experience permanently in mind. The formula for SaaS is *Great User Experience + Great User Interface = Great Product*.

**Performance KPIs in a public cloud—what’s in it for you?**

As we mentioned in the previous section, performance is a tricky subject and, to put it a bit more strongly, if there’s one item that will cause debates in service-level agreements, it’s going to be performance. In terms of KPIs, we need to be absolutely clear about what performance is, in terms of measurable objectives.

What defines performance? It’s the user experience. What about how fast an application responds and processes a request? Note that *fast* is not a measurable unit. A lot of us can probably relate to this: a middle-aged person may think that an app on their phone responding within 10 seconds is fast, while someone younger may be impatiently tapping their phone a second after they’ve clicked on something. They have a relative perception of *fast*. Thus, we need to define and agree on what is measurably fast. One thing to keep in mind is that without availability, there’s nothing to measure, so resilience is still a priority.

What should we measure? Let’s take VMs in the cloud as an example since VMs are still widely used in cloud. Some key metrics that we must take into consideration are:

* **CPU and memory**: All cloud providers offer a wide variety of VM instance sizes. We should look carefully at what instance is advised for specific workloads. For instance, applications that run massive workflow processes in memory require a lot of memory in the first place. For example, SAP S4/HANA instances can require up to 32 GB of RAM or more. For these workloads, Azure, AWS, and GCP offer large, memory-optimized instances that are next to complete SAP solutions. If we have applications that run heavy imaging or rendering processes, we might want to look at specific instances for graphical workloads that use GPUs. So, it comes down to the right type of instance and infrastructure, as well as the sizing. You can’t blame the provider for slow performance if we choose a low-CPU, low-memory machine underneath a heavy-duty application. Use the advisor tools to fulfill the best practices.
* **Responsiveness:** How much time does it take for a server or a service to get a response to a request? There are a lot of factors that determine this. To start with, it’s about the right network configuration, routing, and dependencies in the entire application stack. It does matter whether we connect through a low bandwidth VPN or a high-speed dedicated connection. And it’s also about load balancing. If, during peak times, the load increases, we should be able to deal with that. In the cloud, we can scale out and up, even in a fully automated way. In that case, we need proper configuration for the load balancing solution.
* **IO throughput**: This is about throughput rates on a server or in an environment. Throughput is a measure of **Requests Per Second** (**RPS**), the number of concurrent users, and the utilization of the resources, such as servers, connections, firewalls, and load balancers. One of the key elements in architecture is sizing. From a technological perspective, the solution can be well-designed, but if the sizing isn’t done correctly, then the applications may not perform well or be available at all. The advisor tools that we will discuss in this chapter provide good guidance in terms of setting up an environment, preparing the sizes of the resources, and optimizing the application (code) as such.

The most important thing about defining KPIs for performance is that all stakeholders—business, developers, and administrators—have a mutual understanding of what performance should look like and how it should be measured.

**Optimizing your multi-cloud environment**

Cloud providers offer advisor tools we can use to optimize environments that are hosted on their platforms. In this section, we will briefly look at these different tools and how we can use them.

**Optimizing environments using Azure Advisor**

Like AWS, Azure offers a tool to help optimize environments, called Azure Advisor. Azure Advisor is a service that helps in analyzing our Azure environments and making recommendations around the pillars of the Well-Architected Framework. Next to this, we should evaluate the support plans.

Let’s start with the support plans. Azure offers four types of plans: basic, developer, standard, and professional direct. The basic plan is free, while the most expensive one is professional direct. However, you can’t compare this to the enterprise plan of AWS. Every provider offers free and paid services—the difference per provider is which services are free or must be paid for.

Azure Advisor comes at no extra cost. It provides recommendations on the five pillars of WAF: reliability, security, performance, costs, and operational excellence. The dashboard can be launched from the Azure portal, and it will immediately generate an overview of the status of the resources that we have deployed in Azure, as well as recommendations for improvements. For resiliency, Azure Advisor, for example, checks whether VMs are deployed in an availability set, thereby remediating fault tolerance for VMs. Be aware of the fact that Azure Advisor only advises that the actual remediation needs to be done by the administrator. It offers a recommended action for a recommendation to implement it. A simple interface will open that enables you to implement the recommendation or refer you to documentation that assists you with implementation. We could also automate this with Azure Policy and Azure Automation, but there’s a good reason why Azure Advisor doesn’t already do this. Remediation actions might incur extra costs, and we want to stay in control of our costs. If we automate through Policy and Automation, that’s a business decision and architectural solution that will be included in cost estimations and budgets.

On the other hand, Azure Advisor does provide us with some best practices. In terms of performance, we might be advised to start using managed disks for our app, do a storage redesign, or increase the sizes of our VNets. It’s always up to the administrator to follow up, either through manual tasks or automation.

Next to Advisor, we will use Azure Monitor to guard our resources and Azure Service Health to monitor the status of Azure services. Specific to security monitoring, Azure offers Microsoft Defender for Cloud, previously known as Azure Security Center, and the Azure-native **Security Information and Event Manager** (**SIEM**) tool known as Sentinel. Most of these services are offered on a pay-as-you-go basis: you pay for the ingestion and retention of data that they collect.

More information on Azure Advisor can be found at <https://learn.microsoft.com/en-us/azure/advisor/advisor-overview>.

**Using Trusted Advisor for optimization in AWS**

In all honesty, getting the best out of AWS—or any other cloud platform—is really not that easy. There’s a good reason why these providers have an extensive training and certification program. The possibilities are almost endless, and the portfolios for these cloud providers grow bigger every day. We could use some guidance while configuring our environments. AWS provides that guidance with **Trusted Advisor**. This tool scans your deployments, references them against best practices within AWS, and returns recommendations. It does this for cost optimization, security, performance, fault tolerance, and service limits.

Before we go into a bit more detail, there’s one requirement we must fulfill in order to start using Trusted Advisor: we have to choose a support plan, although a couple of checks are for free, such as a check on **Multi-Factor Authentication** (**MFA**) for root accounts and IAM use. Also, checks for permissions on S3 (storage) buckets are free. Note that basic support is included at all times, including AWS Trusted Advisor on seven core checks, mainly focusing on security. Also, the use of the Personal Health Dashboard is included in basic support.

Support plans come in three levels: developer, business, and enterprise. The latter is the most extensive one and offers full 24/7 support on all checks, reviews, and advice on the so-called Well-Architected Framework, as well as access to AWS support teams. The full service does come at a cost, however. An enterprise that spends one million dollars on AWS every month would be charged around 70 USD per month on this full support plan. This is because AWS typically charges the service against the volumes that a customer has deployed on the platform. The developer and business plans are way lower than that. The developer plan can be a matter of as little as 30 USD per month, just to give you an idea. Note that prices change all the time.

However, this full support does include advice on almost anything that we can deploy in AWS. The most interesting parts, though, are the service limits and performance. The service limits perform checks on volumes and the capacity of a lot of different services. It raises alerts when 80 percent of a limit for that service has been reached, and it then gives advice on ways to remediate this, such as providing larger instances of VMs, increasing bandwidth, or deploying new database clusters. It strongly relates to the performance of the environment: Trusted Advisor checks the high utilization of resources and the impact of this utilization on the performance of those resources.

The full checklist for Trusted Advisor can be found at <https://aws.amazon.com/premiumsupport/technology/trusted-advisor/best-practice-checklist/>.

One more service that we should mention in this section is the free Personal Health Dashboard, which provides us with very valuable information on the status of our resources in AWS. The good news is that not only does it provide alerts when issues occur and impact your resources, but it also guides us through remediation. What’s even better is that the dashboard can give you proactive notifications when planned changes might affect the availability of resources. The dashboard integrates with AWS CloudWatch, but also with third-party tooling such as Splunk, ServiceNow, and Datadog.

**Optimizing GCP with Cloud Trace and Cloud Debugger**

GCP offers two interesting features in terms of optimizing environments: Cloud Trace and Cloud Debugger. Both can be accessed from the portal. From this, you can tell that GCP comes from the cloud-native and native apps world.

Cloud Trace is really an optimization tool: it collects data on latency from the applications that you host on instances in GCP, whether these are VMs, containers, or deployments in App Engine or the native app environment in GCP. Cloud Trace measures the amount of time that elapses between incoming requests from users or other services and the time the request is processed. It also keeps logs and provides analytics so that you can see how performance evolves over a longer period of time. Cloud Trace uses a transaction client that collects data from App Engine, load balancers, and APIs. It gives us good insight into the performance of apps, dependencies between apps, and ways to improve performance.

Cloud Trace doesn’t only work with GCP assets but with non-Google assets too. In other words, we can use Cloud Trace in AWS and Azure as a REST API using JSON.

Cloud Debugger is another tool, and it’s used to debug code in apps that you run in GCP. Debugger will analyze the code while the application is running. It does this by taking a snapshot of the code, although you can use it on the source code as well. It integrates with versioning tools such as GitHub. Debugger supports the most commonly used programming languages in code apps, at least when it runs in containers on GKE. In this case, Java, Python, Go, Node.js, Ruby, PHP, and .NET Core are supported. In Compute Engine, .NET Core is not supported at the time of writing.

Cloud Trace and Cloud Debugger are part of the operations suite of GCP and are a charged service.

More information on Cloud Trace can be found at <https://cloud.google.com/trace/docs/overview/>. Documentation on Cloud Debugger can be found at <https://cloud.google.com/debugger/docs>.

**Optimizing in OCI**

In OCI, we will find a number of tools that will help us in optimizing workloads.

Oracle Cloud Infrastructure Monitoring provides real-time performance metrics, logs, and alerts to help optimize the performance of OCI resources. It monitors the health and performance of resources such as compute instances, load balancers, and databases.

Another tool is Oracle Cloud Infrastructure Advisor, which analyzes OCI resources and provides recommendations to help optimize your infrastructure. It offers suggestions for improving performance, reducing costs, and enhancing security.

Next to this, OCI also offers Oracle Cloud Infrastructure Resource Manager, which helps to automate the creation, deployment, and management of resources. It offers a simple way to manage infrastructure as code and ensure consistency across your environment.

**Use case: creating solutions for business continuity and disaster recovery**

Now that we have gathered the business requirements, identified the risks, and considered our application strategy, including the usage of PaaS and SaaS, we can start thinking about solutions and align these with the requirements. The best way to do this is to create a matrix with the systems, the requirements for resilience, and the chosen technology to get the required resilience. The following table shows a very simple example of this, with purely fictional numbers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **System or system group/category** | **Business level** | **RTO** | **RPO** | **Solution** |
| Application X | Critical | <2 hours | <2 hours | Standby (failover) systems, disaster recovery |
| Application Y | Important | >2 hours <8hours | >2 hours <8 hours | Full daily backup, increments with snapshots |
| Application Z | Non-critical | <48 hours | <48 hours | Weekly backup, daily incrementals |

*Table 7.1: Example of a table for required resilience KPIs*

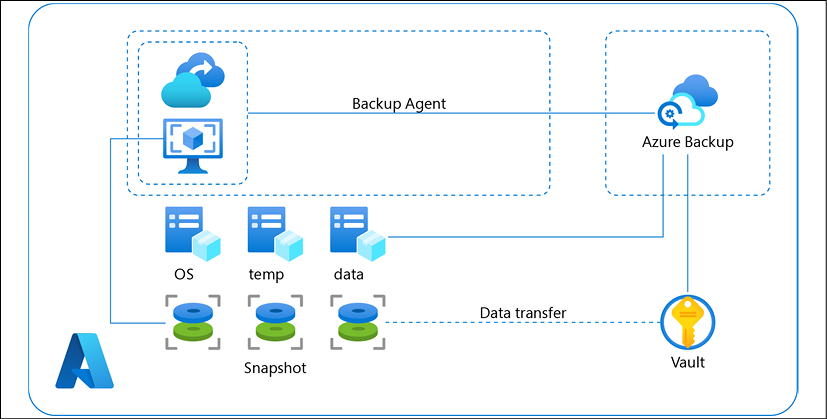
Resilient systems are designed in such a way that they can withstand disruptions. Regardless of how well the systems might be designed and configured, sooner or later, they will be confronted with failures and, possibly, disruptions. Resilience is, therefore, often associated with quality attributes such as redundancy and availability.

**Creating backups in the Azure cloud with Azure Backup and Site Recovery**

Azure Backup works with the principle of snapshots. First, we must define the schedule for running backups. Based on that schedule, Azure will start the backup job. During the initial execution of the job, the backup VM snapshot extension is provisioned on the systems in our environment.

Azure has extensions for both Windows and Linux VMs. These extensions work differently from each other: the Windows snapshot extension works with Windows **Volume Shadow Copy Services** (**VSS**). The extension takes a full copy of the VSS volume. On Linux machines, the backup takes a snapshot of the underlying system files.

Next, we can take backups of the disks attached to the VM. The snapshots are transferred to the backup vault. By default, backups of operating systems and disks are encrypted with Azure Disk Encryption. The following diagram shows the basic setup for the Azure Backup service:



*Figure 7.2: High-level overview of the standard backup components in Azure*

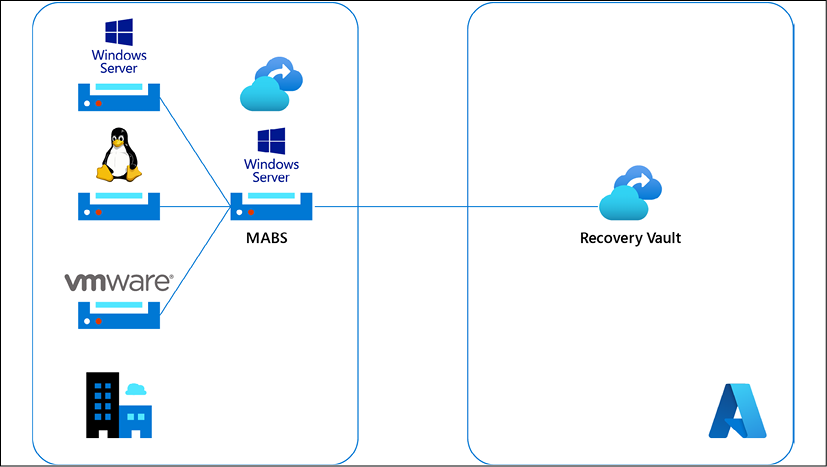
We can create backups of systems that are in Azure, but we can also use Azure Backup for systems that are outside the Azure cloud.

**Backing up non-Azure systems**

Azure Backup can be used to create backups of systems that are not hosted in Azure itself. For that, it uses different solutions. **Microsoft Azure Recovery Services** (**MARS**) is a simple solution to do this. In the Azure portal, we have to create a Recovery Services vault and define the backup goals.

Next, we need to download the vault credentials and the agent installer that must be installed on the on-premises machine or machines that are outside Azure. With the vault credentials, we register the machine and start the backup schemes. A more extensive solution is **Microsoft Azure Backup Server** (**MABS**). MABS is a real VM—running Windows Server 2016 or 2019—that controls the backups within an environment, in and outside Azure. It can execute backups on a lot of different systems, including SQL Server, SharePoint, and Exchange, but also VMware VMs—all from a single console.

MABS, like MARS, uses the recovery vault, but in this case, backups are stored in a geo-redundant setup by default. The storage replication option allows you to choose between geo-redundant storage and locally redundant storage. The following diagram shows the setup of MABS:



*Figure 7.3: High-level overview of the setup for Microsoft Azure Backup Server*

Documentation on the different backup services that Azure provides can be found at <https://docs.microsoft.com/en-us/azure/backup/>.

Before we dive into the recovery solutions for Azure and the other cloud providers, we will discuss the generic process of disaster recovery. Disaster recovery has three stages: detect, response, and restore. We must have monitoring in place that is able to detect whether critical systems are failing and that they are not available anymore.

It then needs to trigger actions to execute mitigating actions, such as failover to standby systems that can take over the desired functionality and ensure that business continuity is safeguarded. The last step is to restore the systems back to the state that they were in before the failure occurred. In this last step, we also need to make sure that the systems are not damaged in such a way that they can’t be restored.

Recovery is a crucial element in this process. However, recovery can mean that systems are completely restored back to their original state where they still were fully operational, but we can also have a partial recovery where only the critical services are restored, and, for example, the redundancy of these systems must be fixed at a later stage. Two more options are **cold standby** and **warm standby**. With cold standby, we will have systems that are reserved that we can spin up when we need them. Until that moment, these systems are in shut down modus. In warm standby, the systems are running but not yet operational in production modus. Warm standby servers are much faster to get operational than cold standby servers, which merely have reserved capacity available.

Donald Firesmith wrote an excellent blog post about resilience for the Software Engineering Institute of Carnegie Mellon University. You can find it at [https://insights.sei.cmu.edu/sei\_blog/2019/11/system-resilience-what-exactly-is-it.html](https://insights.sei.cmu.edu/blog/system-resilience-what-exactly-is-it/).

**Understanding Azure Site Recovery**

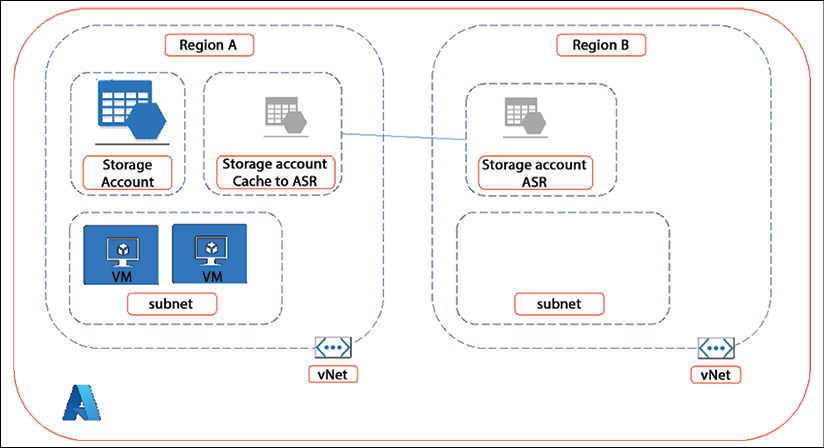
**Azure Site Recovery** (**ASR**) offers a solution that helps set up disaster recovery in Azure. In essence, ASR replicates and orchestrates the replication of VMs (between regions in Azure, on-premises to Azure, or on-premises to secondary on-premises). If the primary location where you host the environments becomes unavailable because of an outage, ASR will execute a failover to the secondary location, where the copies of your systems are. As soon as the primary location is back online again, ASR will execute the failback to that location again.

A bit of bad news is that this is not as simple as it sounds. You will need to design a recovery plan and assess whether workloads can actually failover from the application and data layer. Then, probably the trickiest part is getting the network and boundary security parameters right: think of switching routes, reserved IP addressing, DNS, and replicating firewall rules.

Azure has solutions for this as well, such as DNS routing with traffic manager, which helps with DNS switching in case of a failover, but still, it takes some engineering and testing to get this in place.

The last thing that really needs serious consideration is what region you will have the secondary location in. A lot of Azure regions do have dual zones (data centers), but there are some regions that only have one zone, and you will need to choose another region for failover. Be sure that you are still compliant in that case. To ensure resiliency, a minimum of three separate availability zones are present in all availability zone-enabled regions.

The following diagram shows the basic concept of ASR. It’s important to remember that we need to set up a cache storage account in the source environment. During replication, changes that are made to the VM are stored in the cache before being sent to storage in the replication environment:



*Figure 7.4: High-level overview of ASR*

More information on ASR can be found at <https://docs.microsoft.com/en-us/azure/site-recovery/>.

With that, we have covered Azure. In the next section, we will look at backup and disaster recovery in AWS and GCP.

**Working with AWS backup and disaster recovery**

In this section, we will explore the backup and disaster recovery solutions in AWS. We will learn how to create backups based on policies and on tags. We will also look at the hybrid solution for AWS.

**Creating policy-based backup plans**

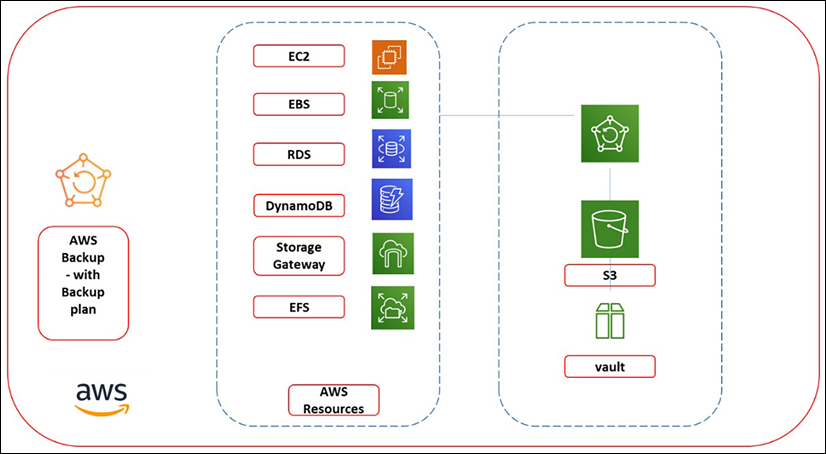
As in Azure, this starts with creating a backup plan that comprises backup rules, frequency, windows, and defining and creating the backup vault and the destination where the backups should be sent. The backup vault is crucial in the whole setup: it’s the place where the backups are organized and where the backup rules are stored. You can also define the encryption key in the vault that will be used to encrypt the backups. The keys themselves are created with AWS **Key Management Service** (**KMS**). AWS provides a vault by default, but enterprises can set up their own vaults.

With this, we have defined a backup plan, known as backup policies in AWS. These policies can now be applied to resources in AWS. For each group of resources, you can define a backup policy in order to meet the specific business requirements for those resources. Once we have defined a backup plan or policy and we have created a vault, we can start assigning resources to the corresponding plan. Resources can be from any AWS service, such as EC2 compute, DynamoDB tables, **Elastic Block Store (EBS)** storage volumes, **Elastic File System (EFS)** folders, **Relational Database Services (RDS)** instances, and storage gateway volumes.

**Creating tag-based backup plans**

Applying backup plans or policies to resources in AWS can be done by simply tagging the plans and the resources. This integration with tags makes it possible to organize the resources and have the appropriate backup plan applied to these resources. Any resource with a specific tag will then be assigned to the corresponding backup plan. An example, if we have set out policies for business-critical resources, we can define a tag that says BusinessCritical as a parameter for classifying these resources. If we have defined a backup plan for BusinessCritical, every resource with that tag will be assigned to that backup plan.

The following diagram shows the basic concept of AWS Backup:



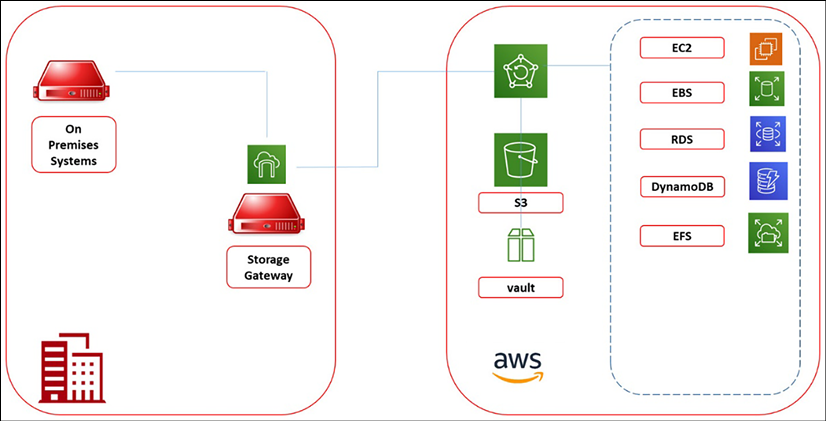
*Figure 7.5: High-level overview of AWS Backup*

Similar to Azure, we can also create backups of systems that are outside of AWS using the hybrid backup solution of AWS. We’ll describe this in the next section.

**Hybrid backup in AWS**

AWS calls backing up resources in AWS a native backup, but the solution can be used for on-premises workloads too. This is what AWS calls hybrid backup. For this, we’ll have to work with the AWS Storage Gateway. We can compare this to MABS, which Microsoft offers.

In essence, the on-premises systems are connected to a physical or virtual appliance over industry-standard storage protocols such as **Network File System** (**NFS**), **Server Message Block** (**SMB**), and **internet Small Computer System Interface** (**iSCSI**). The appliance—the storage gateway—connects to the AWS S3 cloud storage, where backups can be stored. You can use the same backup plans for hybrid backup that you do for the native backup. The following diagram shows the principle of hybrid backup:



*Figure 7.6: High-level overview of hybrid backup in AWS*

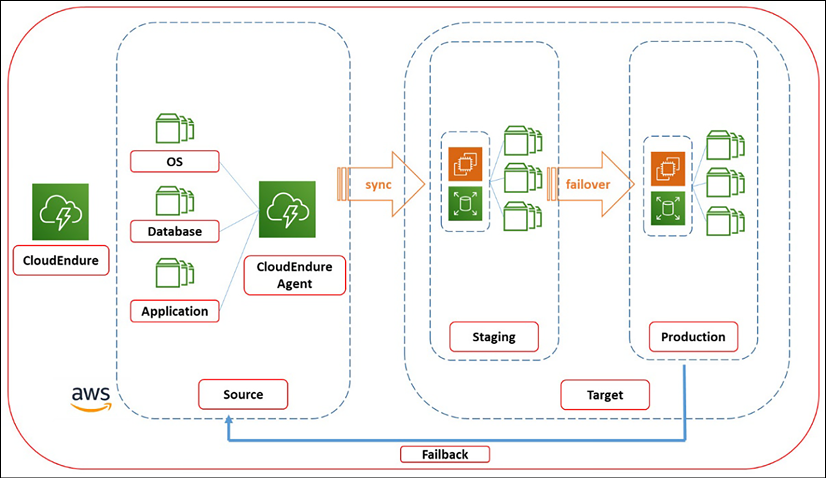
Now, let’s learn about the disaster recovery options available in AWS.

**AWS disaster recovery and cross-region backup**

AWS allows us to perform cross-region backups, meaning that we can make backups according to our backup plans and replicate these to multiple AWS regions. However, this occurs in the data layer. We can do this for any data service in AWS like RDS, EFS, EBS, and Storage Gateway volumes. So, with Storage Gateway included, it also works for data that is backed up on-premises.

Next to this, AWS also has another proposition that’s a **Business Continuity and Disaster Recovery** (**BCDR**) solution: CloudEndure **disaster recovery** (**DR**). This solution doesn’t work with snapshots but keeps target systems for DR continuously in sync with the source systems with continuous data protection. By doing this, they can even achieve sub-second recovery points and barely lose any data. CloudEndure supports a lot of different systems, including physical and virtualized machines, regardless of the hypervisor. It also supports enterprise applications such as Oracle and SAP.

This principle is shown in the following diagram:



*Figure 7.7: High-level overview of the CloudEndure concept in AWS*

CloudEndure uses agents on the source systems and a staging area in AWS where the system duplicates are stored on low-cost instances. In the case of a failover, the target DR systems are booted and synced from this staging area. The failback is done from the DR systems in AWS.

More information on AWS Backup can be found at <https://docs.aws.amazon.com/aws-backup/latest/devguide/whatisbackup.html>. Documentation on CloudEndure can be found at <https://aws.amazon.com/cloudendure-disaster-recovery/>.

**Creating backup plans in GCP**

GCP also uses snapshot technology to execute backups. The first snapshot is a full backup, while the ones that follow are iterative and only back up the changes that have been made since the last snapshot. If we want to make a backup of our data in GCP Compute Engine, we have to create a persistent disk snapshot. It’s possible to replicate data to a persistent disk in another zone or region, thus creating geo-redundancy and a more robust solution.

As with AWS and Azure, you will first have to design a backup plan, or in GCP, a snapshot schedule so that backups are taken at a base frequency. Next, we have to set the storage location. By default, GCP chooses the region that is closest to the source data. If we want to define a solution with higher availability, we will need to choose another region ourselves where we wish to store the persistent disks.

Be aware that, in GCP, we work with constraints. If we have defined a policy with a constraint that data can’t be stored outside a certain region, and we do pick a region outside of that boundary, the policy will prevent the backup from running.

GCP proposes to flush the disk buffers prior to the backup as a best practice. You don’t need to stop the applications before snapshots are taken, but GCP does recommend this so that the application stops writing data to disk. If we stop the application, we can flush the disk buffer and sync the files before the snapshot is created. For Unix programmers, this will be very familiar since GCP lets you connect with SSH to the disk and sudo sync to execute the synchronization process. All of this is done through the command-line interface.

But what about Windows? We can run Windows-based systems in GCP, and we can take backups of these systems. GCP uses VSS for this, which is the Volume Shadow Copy Services of Windows. Before we do that, GCP recommends unmounting the filesystem and then taking the snapshots. We can use PowerShell for this.

More information on backups in Compute Engine from GCP can be found at <https://cloud.google.com/compute/docs/disks/create-snapshots>. Specific documentation and how-to information about creating snapshots of Windows systems can be found at <https://cloud.google.com/compute/docs/instances/windows/creating-windows-persistent-disk-snapshot>.

**Disaster recovery planning**

GCP lets us define the RTO and RPO first when planning a DR strategy. Based on the requirements, we define the building blocks within GCP to fulfill the strategy. These building blocks comprise Compute Engine and Cloud Storage. In Compute Engine, we can define how we want our VMs to be deployed and protected from failures. The key components here are persistent disks, live migration, and virtual disk import. We discussed the creation of persistent disks as part of backing up in GCP. Live migration keeps VMs up by migrating them into a running state to another host. Virtual Disk Import lets you import disks from any type of VM to create new VMs in Compute Engine.

These newly created machines will then have the same configuration as the original machines. The supported formats are **Virtual Machine Disk** (**VMDK**), which is issued by VMware, and **Virtual Hard Drive** (**VHD**), which is used by Microsoft and RAW.

As you can tell, GCP does not offer a predefined solution for DR. And again, there’s much more focus on containers with GKE. GKE has some built-in features that you can use to create highly available clusters for containers. Node auto-repair checks the health status of cluster nodes: if nodes don’t respond within 10 minutes, they get automatically remediated. If we’re running nodes in a multi-region setup, GKE offers multi-cluster ingress for Anthos as a load-balancing solution between the clusters. This solution was previously offered as **Kubernetes Multicuster Ingress** (**Kubemi**). For all of this, we do need a solution to route the traffic across the GCP regions and to make sure that DNS is pointing to the right environments. This is done through Cloud DNS using Anycast, the Google global network, and Traffic Director.

More information on disaster recovery planning in GCP can be found at <https://cloud.google.com/solutions/dr-scenarios-planning-guide>.

Google suggests looking at third-party tools when we have to set up a more complex DR infrastructure solution. Some of the tools that are mentioned in the documentation include Ansible with Spinnaker and Zero-to-Deploy with Chef on Google Cloud.

**Creating backups in OCI**

OCI offers several options for creating backups, including the Backup service, Database backup, Object Storage, and Compute Instance backup.

The first solution to look at is the OCI Backup service, as a fully managed, centralized backup solution for all resources that we host in OCI. It offers automatic backup policies and retention policies and allows you to create on-demand backups of resources. The service supports backups for a variety of OCI resources, including compute instances, block volumes, file storage, and database services.

We can use OCI Database backup specific for the databases that we run in OCI. This is a built-in backup and recovery feature. The service offers several backup types, including full, incremental, and automated backups, and allows us to schedule backups and retention policies. The backup files are stored in OCI Object Storage. Restoring a database can be done to any point in time within the backup retention period.

If we have deployed OCI Compute instances, we can create manual backups using the OCI Console or the OCI CLI. The backup creates a boot volume backup, which includes the instance’s boot volume and any attached Block Volumes. Now, we can restore the backup to a new instance or to the same instance and restore the entire backup or individual files.

More information on backups and disaster recovery in OCI can be found at <https://www.oracle.com/database/technologies/high-availability/secure-backup.html>. Specific details on implementing a DR strategy in OCI can be found at <https://www.oracle.com/cloud/backup-and-disaster-recovery/>.

We have discussed backup solutions from the cloud provider themselves. There’s a risk in doing this: we are making our businesses completely dependent on the tools of these providers. From an integration perspective, this may be fine, but a lot of companies prefer to have their backups and DR technology delivered through third-party tooling. Reasons to do this can sprout from compliance obligations but also from a technological perspective. Some of these third-party tools are really specialized in these types of enterprise cloud backup solutions, can handle many more different types of systems and data, and can be truly cloud agnostic. Examples of such third-party tools include Cohesity, Rubrik, Commvault, and Veeam.

The most important thing is to have a good strategy. This starts with defining the RPO and RTO: RPO and RTO are critical metrics that define how much data you can afford to lose and how quickly you need to recover your services. These metrics are specific to your business needs and should be defined before designing your DR strategy. Possible strategies are:

* Active-Active: In an active-active configuration, you have multiple active environments in different regions. This configuration provides high availability and can be used to distribute traffic across regions. However, it can be more complex to manage than other configurations.
* Active-Passive: In an active-passive configuration, you have a primary environment in one region and a secondary environment in another region. The secondary environment is only activated in the event of a disaster. This configuration is simpler to manage but may have longer RTOs.

Lastly, we need to replicate the data and design the failover tactic, including the DNS failover, in case we need to execute our DR strategy.

This concludes the chapter where we have learned to architect and deploy applications to the cloud, making use of the various native services that cloud providers offer to host these applications and making sure that applications perform well and are resilient. In the next chapter, we will discuss the data in cloud.

**Summary**

In this chapter, we discussed the definitions of resilience and performance. Companies want their applications to be available and to perform well: it defines the customer experience. We discussed various concepts that developers can use to build performant and resilient applications, using cloud-native tools as much as possible in 12-factor apps, PaaS, and SaaS.

We also learned how to optimize our environments using different advisory tools that cloud providers offer. We then learned how to identify risks in the various layers: business, data, applications, and technology. We studied the various methods we can use to mitigate these risks.

One of the biggest risks is that we “lose” systems without the ability to retrieve data from backups or without the possibility of failover to other systems. For real business-critical systems, we might want to have disaster recovery, but at a minimum, we need to have proper backup solutions in place. Various backup and disaster recovery solutions that are available in the major cloud platforms have been explored.

We talked about infrastructure and applications a lot in the past chapters. But applications only make sense if there’s data that these applications can use. In a connected world, the amount of data that is available to us is massive. How do we design platforms that can handle these amounts of data and make it usable for specific business goals? The next chapter is all about data platforms in the cloud.

**Questions**

1. What do the terms RPO and RTO stand for?
2. What tool would you use to capture failures in application code that’s running in Google Cloud?
3. Cloud providers offer toolkits to develop SaaS applications. Name the service that AWS offers to build and host SaaS applications.
4. True or false: We can use the backup solutions in Azure and AWS for systems that are hosted on-premises too.

**Further reading**

* *Reliability and Resilience on AWS*, by Alan Rodrigues, Packt Publishing
* *Architecting for High Availability on Azure,* by Rajkumar Balakrishan, Packt Publishing

**Join us on Discord!**

Read this book alongside other users, cloud experts, authors, and like-minded professionals.Ask questions, provide solutions to other readers, chat with the authors via. Ask Me Anything sessions and much more.

Scan the QR code or visit the link to join the community now.

<https://packt.link/cloudanddevops>