**Introduction**

Cloud native development has become the de-facto architecture of choice for newly built applications. Cloud native development gives enterprises the capability to fully realize the advantages of leveraging a cloud platform and enables quick iteration and portability. Cloud native development takes a different and modern approach to designing, building, deploying, and managing applications. This approach places automation, elasticity, and resiliency front and center by leveraging cloud platforms such as Oracle Cloud Infrastructure (OCI).

OCI is a next-generation cloud designed to run any application faster and more securely, for less. It is one of the fastest-growing cloud platforms, in terms of customer growth and global geographical footprint. OCI includes tools and utilities for building new cloud native applications and also running existing enterprise applications without rearchitecting them. OCI is built around the pillars of AI and autonomous systems, enterprise security, and open standards.

**Goals and Approach**

Whether you are new to the cloud paradigm or new to just OCI, this book aims to give you a complete rundown of the services in OCI that help you build cloud native applications. Because lines between infrastructure engineers and developers are blurring, the book covers both infrastructure services and developer services, as well as how to manage them in the context of a cloud native application development environment.

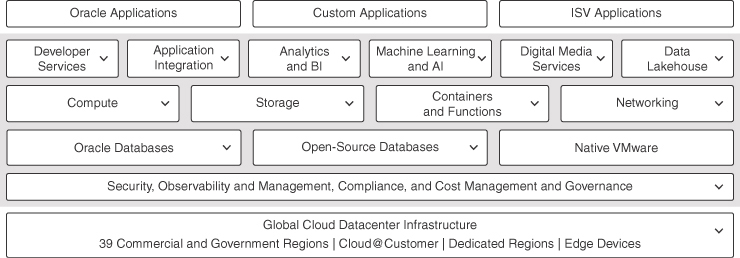
## 1

## Introduction to Oracle Cloud Infrastructure

Oracle Cloud Infrastructure (OCI) offers a comprehensive platform of public cloud services that enable enterprises, Independent Software Vendors (ISVs), and startups to create cloud-scale solutions that are secure, highly available, and geographically distributed on one of the fastest-growing cloud provider footprints.

Cloud adoption can empower your organization to improve business agility and promote innovative solutions. However, every cloud infrastructure platform uses its own architecture and terminology. This makes the process of getting started on a cloud platform equally challenging for both new users and users who have worked on other cloud platforms. Understanding these terms also helps you understand how OCI is distributed across the globe, how it organizes resources, and how it secures access to them. If you are new to working on a cloud platform, this section introduces the basic OCI terminology. On the other hand, if you are familiar with other cloud platforms, many of these terms and concepts might sound familiar; however, they might also differ in small but important ways when compared to similar terms or concepts in other cloud platforms.

The OCI service portfolio is broad, encompassing infrastructure services, security and identity services, developer platforms, analytics platforms, machine learning and artificial intelligence (AI) platforms, media services, and more. [Figure 1-1](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01.xhtml#ch01fig01) shows a high-level overview of the various classes of services that are available in OCI. This book focuses on building cloud-native applications and discusses the services and platforms in OCI that help you build modern, distributed, and resilient applications. These include services such as Oracle Container Engine for Kubernetes (OKE), Oracle Container registry, OCI messaging and observability platforms, OCI service mesh, API gateways, and more. These services build on the foundational concepts of how resources are distributed, organized, secured. This chapter covers these foundational concepts in OCI and introduces its vocabulary. To get the most from OCI’s services, this chapter also presents a set of best practices for effectively planning and executing your OCI adoption.



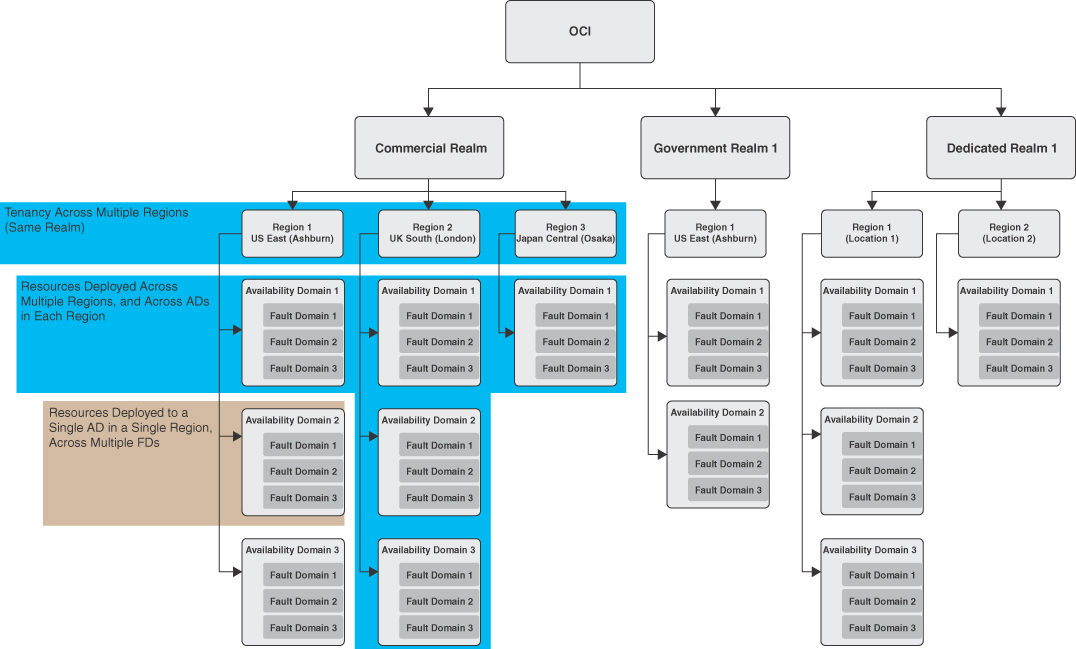
**Figure 1-1** A High-Level Overview of the Services OCI Offers

### Realms, Regions, and Availability Domains

OCI as a cloud platform is organized into multiple realms. An example of a realm, and the most common realm for most users of OCI, is the OCI commercial realm. Most users do not even realize that realms exist, because they most often interact with just the commercial realm. A realm is a logical construct that is spread across geographies and physical data centers. Realms are completely isolated from each other, even when they might share a physical location. Examples of other realms include the government realms for the United States, the United Kingdom, and more. These realms have a different geographical spread than the commercial realm.

A realm is made up of one or more regions. Regions are geographical areas around the globe where OCI has a presence. At the time of writing, the OCI commercial realm is spread across more than 45 regions across the globe and is used by OCI commercial customers (see [Figure 1-2](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01.xhtml#ch01fig02)). The OCI approach to regions is to have regions close to customers. The OCI fundamental design and architecture prioritizes the speed and efficiency of launching new regions, allowing OCI to rapidly scale its footprint across the globe. The OCI focus on enterprise workloads prioritizes data sovereignty and business continuity requirements for customers, which drives its strategy of building multiple cloud regions in every country.

Regions themselves are a logical grouping of one or more availability domains, which are physical data centers located within the same relative geographical area. Some regions have multiple availability domains, which means that the region has multiple physical data centers that are connected by a fully encrypted low-latency/high-bandwidth network. Cloud resources can be regional in nature, as with a Kubernetes cluster, or they can be specific to an availability domain, as with a node (compute instance) within that cluster. [Figure 1-3](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01.xhtml#ch01fig03) shows the OCI region presence at the time of writing.



**Figure 1-2** OCI Realms, Regions, and Availability Domains



**Figure 1-3** Oracle Cloud Infrastructure Global Footprint

An availability domain offers a construct known as a fault domain for applications to implement fault tolerance and high availability. Every availability domain has three fault domains that offer completely isolated physical hardware grouping. Each fault domain has its own hardware resources, including power distribution units, cooling, and more, within a single availability domain. This enables you to ensure that redundant resources you use are isolated at a hardware level, protecting your workloads from hardware failure and maintenance outages. Depending on the type of workloads and their characteristics, fault tolerance and high availability can be implemented within a single availability domain, spanning multiple availability domains within a region or even spanning the workload across multiple regions.

### Tenancies and Compartments

A tenancy is a partition within an OCI realm that identifies a single customer subscription. A tenancy is created when you subscribe to OCI and is secure and isolated from other tenancies. A tenancy can subscribe to any region that is part of its realm and use the resources from any of the subscribed regions. It also acts as the top aggregate unit for cost management and budgeting tools that OCI provides out of the box. These included tools, such as the tenancy explorer, can aggregate resource use across regions to give users insights into their tenancy’s resource use across all geographies.

Most organizations use a single tenancy and usually want to have a management model that is similar to their existing processes. This generally means that they carve up the tenancy and give individual departments or business units more autonomous control over their own collections of resources; the global infrastructure team then sets standards on usage quotas, security posture, and more. In other words, administrators usually compartmentalize their workloads or business units. OCI offers a construct for this exact purpose. Within a tenancy, you can organize your resources into compartments. Compartments are not just for organizing resources; they also let you manage who has access to perform various operations on resources within a compartment. This lets administrators create policies that govern how an organization’s members access and interact with the resources. Compartments can be hierarchical: They can have nested compartments up to six levels deep. The root of all compartments is tenancy itself. As a simple example, an administrator could designate one group of users to have only read-only access to resources within a compartment, grant another group access to modify certain resources, and allow yet another group complete control, including the capability to create and terminate resources. Although this model is common, other models are also possible, depending on an organization’s structure and what is conducive for the users. For instance, a large organization might operate as a set of independent business units that own their individual tenancies. Likewise, another organization could be horizontally integrated, with multiple shared databases colocated in a compartment that is managed by a team of database administrators (DBAs).

A cloud resource, or simply a resource, is any service or object that users create in OCI and that represents some capability. For instance, a Kubernetes cluster, a compute instance, a network security group, and a user account are all examples of resources. The various services that make up OCI offer APIs and tooling built on the APIs to enable users to interact with and manipulate these resources. Oracle assigns a unique identifier called an Oracle Cloud Identifier (OCID) to all resources. Users can use OCIDs to identify and operate on specific resources when using the OCI API, the command-line interface (CLI), or tools such as Terraform. OCIDs are structured as follows:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01_images.xhtml#f005-01a)

ocid1.<RESOURCE TYPE>.<REALM>.[REGION][.FUTURE USE].<UNIQUE ID>

An example OCID for a block volume could look like ocid1.volume.oc1.iad.xxxx[truncated]xxxx and an example OCID for a User could look like ocid1.user.oc1..xxxx[truncated]xxxx. Note that the region identifier in an OCID is optional and is present only for region-specific resources such as block volumes or compute instances (they are always located in a specific region). Global resources, such as a user, do not have this identifier because the user is not localized to any region but instead is valid across all regions to which the tenancy has subscribed.

### Controlling Access to Resources

A common challenge for infrastructure teams is figuring out how to give their users a certain amount of autonomy without compromising their security posture. This is more challenging than it sounds. It starts with the task of identifying users, who can be human or nonhuman. The users can be transient because team members are generally added and removed or because a particular person’s access changes over time. General industry principles such as the principle of least access promote an access pattern in which users are given the least privileges to accomplish their tasks and are restricted to the most specific resources for the shortest duration of time. This presents several challenges, such as needing to be very specific about the resources that a user can effect and further narrowing access to specific types of operations on those resources that are allowed and specific time frames.

To address these needs, OCI provides a built-in identity and access management (IAM) capability. OCI IAM enables you to manage access to your cloud resources in a declarative manner that spans regions. IAM provides resources such as users, groups, and policies. A user resource in OCI represents a subject that authenticates with OCI and works with OCI resources. A group resource in OCI is simply made up of zero or more users, making it simple to manage access for a collection of users by treating them as a logical collection. A user can be a part of multiple groups. IAM uses policies to describe access control, and it confers privileges to groups, never individual users. A user starts out as not a member of any group and, hence, has no privileges (other than perhaps changing passwords). When a user is added to a group, the user is conferred the privileges that the policy has granted to the group; if a user is a member of multiple groups, the user gets the widest permissions that are allowed by the totality of the group memberships.

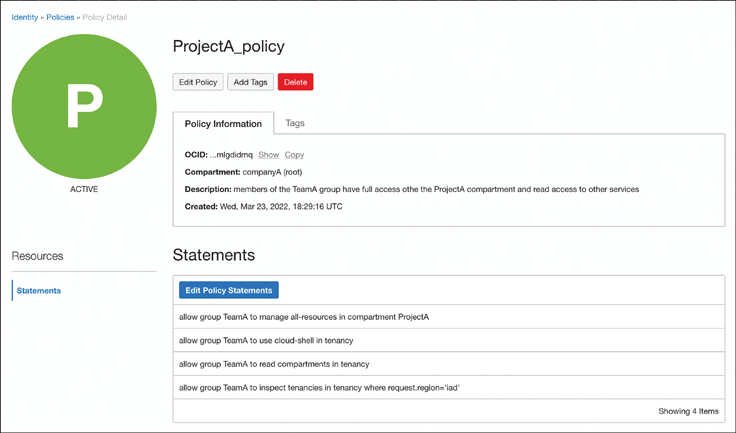
Policies are designed to be intuitive and human readable, which makes it easy to create and maintain policies over time. Policies follow this structure:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01_images.xhtml#f006-01a)

Allow group <group\_name> to <verb> <resource-type> in compartment

<compartment\_name>

[Figure 1-4](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01.xhtml#ch01fig04) shows a policy made up of several policy statements.



**Figure 1-4** A Policy Document Consisting of Intuitive Policy Statements

Notice that the policy statements always start with the word allow. By default, there are no permissions; all policies add permissions and never take away permissions. In a policy statement, you can refer to group names using their OCIDs or names, and you can include multiple groups (comma separated) within the same statement.

The verbs indicate what types of operations are granted by the policy. Verbs can be inspect, read, use, and manage. Both inspect and read verbs allow nonmutating operations and essentially grant read-only access. The difference between them is that inspect restricts the amount of data you can read about the resource, whereas read allows the user to get complete metadata about a resource. The use verb grants use of the resource and limited modifications to resources, but it limits this to existing resources; it does not include the ability to create or delete resources. Furthermore, the use verb does not cover modifications that are equivalent to creation and deletion.

The various verbs attempt to aggregate API operations for each service in a generic manner. Ultimately, OCI is a cloud platform, and every resource provided by the platform is manipulated through the APIs that it exposes. The types of operations that are available for each resource differ, so the various verbs for each resource can also have different meanings. For instance, consider the verb use in the context of networking and in the context of storage: They could mean different things. Therefore, OCI documentation[**1**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01.xhtml#ref1_1a) provides a complete reference for each service and how the various verbs affect each resource. The various verbs and their effects in general terms can be summarized as follows:

* **inspect:** Fetch a limited amount of data about a resource
* **read:** Fetch the complete metadata for a resource
* **use:** Interact with the resource, without being able to create or delete it
* **manage:** Operate with complete control over the resource

To illustrate these verbs in action, consider the act of updating the Kubernetes version used by a Kubernetes cluster running on Container Engine for Kubernetes. This is not covered by the use verb and requires the permissions conferred by the manage verb. On the contrary, consider the act of generating a kubeconfig file. The cluster exists, and generating a kubeconfig file does not modify the cluster in any way. Therefore, a user who has permissions to use clusters can request the configuration required to connect to it. The manage verb, on the other hand, confers permissions to perform any action on the resource, including upgrading it to a newer version, creating new clusters, and terminating existing ones.

The verb in a policy statement is followed by the resource type. This can be a very specific resource or a more generalized family of related resources. Although it is good practice to individually identify the resources the policy affects, many times resources are closely related and often managed together. In these cases, a single policy statement can refer to a family of resources that the policy statement will affect equally. The primary advantage of using family resource types is the capability to write fewer policies and ease policy management. The downside is that if OCI adds a new resource to a family, the existing policies will automatically apply to the new resource, which is sometimes undesirable. As an example, consider the following resource types:

* clusters**:** Represent Kubernetes clusters running on Oracle Container Engine for Kubernetes.
* cluster-node-pools**:** Represent a pool of nodes that act as worker nodes for deploying and running workloads in this cluster. A cluster may have many node pools.
* cluster-work-requests**:** Consist of resources that track long-running tasks, such as upgrading your cluster or scaling your node pool, and other cluster management operations that are executed asynchronously.

These three resource types are related to each other and can be collectively addressed as the cluster-family aggregate resource type. In the future, if another resource type that is closely related to these is added, the cluster-family aggregate resource type could encompass that resource type as well. In other words, the aggregate resource type’s scope can change as the services expand and evolve.

Using these resource types individually, we can craft policies that give various groups access to very specific operations. This approach might be well suited for organizations that have clear responsibility boundaries or are required to isolate access for regulatory reasons. Such an organization might choose to use policy statements as follows:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01_images.xhtml#f008-01a)

Allow group k8s\_admin to manage clusters in compartment project\_A

Allow group k8s\_admin to manage cluster-node-pools in compartment project\_A

Allow group k8s\_admin to manage cluster-work-requests in compartment project\_A

Allow group k8s\_ops to read clusters in compartment project\_A

Allow group k8s\_ops to read cluster-node-pools in compartment project\_A

Allow group k8s\_ops to read cluster-work-requests in compartment project\_A

These policy statements identify two groups, k8s\_admin and k8s\_ops. The k8s\_admin group has full control over these resources, whereas the k8s\_ops group has the ability to read the resources and their metadata. Note that both groups have access to the same resource types, but their level of access or the operations that members of each group are allowed to perform are different. Using the aggregate resource type, the same policy statements can be rewritten as follows:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01_images.xhtml#f008-02a)

Allow group k8s\_admin to manage cluster-family in compartment project\_A

Allow group k8s\_ops to read cluster-family in compartment project\_A

An agile team in which developers take on multiple roles might find it easier to create relatively coarse-grained policies that permit the same level of access to all the resource types within the family. It is important to note that, when using an aggregate resource type such as cluster-family, if the aggregate resource type adds a new individual resource type to it, these policies will now grant access to that new resource type, which did not exist at the time the policy was written. Therefore, you should take care when using aggregate resource types: Over time, the policy could potentially give users access to more resources (but always closely related ones) than at the time of its writing. Being more specific might be verbose, but it is usually more precise and consistent with the intent. The contrary view to this approach could be that an aggregate would only ever add resources that are closely related and, therefore, if a closely related new resource type is added in the future, it might invariably trigger an updating of the policy to include the new resource type. The choice here depends on your preference, established organizational processes, and security posture.

The final part of the preceding policy syntax is the scope of the policy statement—in other words, where and at what level this policy should be applied. This is typically a compartment. In the preceding example, the permissions are granted to the groups k8s\_admin and k8s\_ops for all resource types in the cluster-family in the compartment named ProjectA. This keeps the policies you write simple to read, understand, and maintain, yet keeps them flexible enough that you can implement an access pattern to give users the least amount of privileges to accomplish their tasks to the most specific resources. The privileges granted by a policy statement are limited to classes of operations (verbs) and scoped to groups of users. The resources to which the access is granted are limited to the resource types and scoped to the compartment specified. When an authenticated user attempts to perform an operation on OCI—regardless of whether the API is used directly or a higher-level abstraction is used (such as the Software Development Kits [SDKs], CLI, Terraform, Ansible, or the web console)—the IAM system checks whether the user (or, more specifically, any group that the user belongs to) is allowed to perform the requested operation.

Policies also offer advanced syntax elements that are a lot more granular and nuanced than the simple syntax you have seen so far. As part of a policy statement, you can specify one or more conditions that must be met for access to be granted. These conditions are based on variables that represent either the access request itself or the resource on which the action is taken. The variable that encapsulates the request parameters and characteristics is named request. The variable that represents the resource being acted upon is named target. Using these variables enables you to make policies a lot more fine-grained. As an example, consider the following policy:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01_images.xhtml#f009-01a)

Allow group vendor-admins to manage clusters in tenancy

where all { request.utc-timestamp.time-of-day between '17:00:00Z' and

'01:00:00Z',

target.compartment.id != 'OCID\_for\_production\_compartment',

request.permission != ‘CLUSTER\_DELETE’}

This policy enables members of the group vendor-admins to manage clusters in the tenancy if they meet some conditions. The conditions are combined with the all keyword, indicating that all the conditions need to be met if access is to be granted. First, the group is granted the permission only when the requests are made during a specific time interval of the day. Second, the users are allowed to manage clusters in any compartment except the production compartment. Last, the CLUSTER\_DELETE permission is not granted. This uses the notion of permission, which allows you to be very specific in your policy statements. Permissions represent the operations that a user is allowed to perform on a resource. These operations depend on the resource itself. Verbs such as use and manage, in the context of a specific service, are simply bundles of these permissions. The Oracle documentation comprehensively covers these advanced constructs.[**2**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01.xhtml#ref1_2a)

Similar to groups, OCI supports a construct called a dynamic group. Dynamic groups are defined in terms of a set of matching rules. All resources that match the rules are considered members of the group. For instance, consider a dynamic group named build-nodes that is defined by a matching rule that matches all instances in a specific compartment named build-nodes. The matching rule might look as follows:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01_images.xhtml#f009-02a)

instance.compartment.id = '<compartment\_ocid for the build-nodes compartment>'

Now consider the following policy:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01_images.xhtml#f010-01a)

allow dynamic-group build-nodes to use devops-repository in compartment

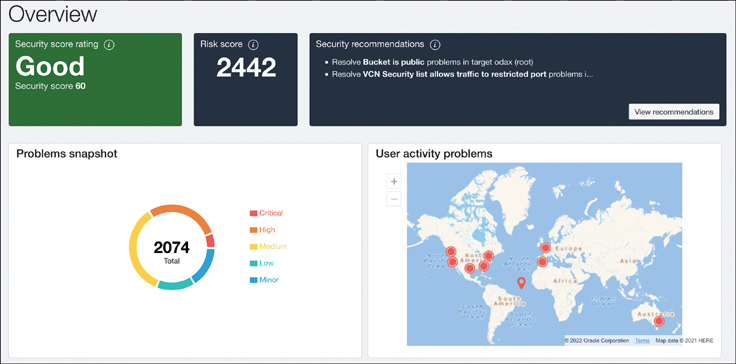
Project-A

Any instance created in the compartment build-nodes is automatically a member of the dynamic group. When the instance in this compartment tries to access a devops-repository in the Project-A compartment, its permissions are checked and IAM will resolve it as having access. This access is granted because the instance that is requesting access is part of the dynamic group build-nodes and the dynamic group is given access through the policy.

### Cloud Guard and Security Zones

Cloud Guard is an OCI service that constantly examines your OCI resources, looking for potential security issues, configurations that increase the attack surface, or user activity that deviates from normal. When Cloud Guard detects a problem, it can alert you or take corrective actions itself. The general workflow in Cloud Guard is to define a scope for the resources to target, usually a compartment. When that is set, Cloud Guard continues to monitor the resources it is targeting, using a set of detectors. The detectors’ behavior is encapsulated in a set of rules bundled as a recipe. When rule violations are detected, it raises a problem. Responders then can take an appropriate action, based on rules defined in a responder recipe. This flow and behavior are fully customizable. You can use Oracle-provided detector recipes to watch for issues or build your own ruleset as a recipe. Similarly, the actions that are taken can be customized by either using the Oracle-provided recipes or building your own. [Figure 1-5](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01.xhtml#ch01fig05) shows the Cloud Guard dashboard that organizes the high-level report from Cloud Guard, enabling you to drill down into the problems that are detected.

Even though Cloud Guard constantly watches over its list of targets using the detectors you specify, it can be useful to set some ground rules to prevent policy violations instead of simply detecting and remediating them. Security zones provide exactly this capability. Security zones work by validating operations, such as creating or updating resources in real time to deny operations that would result in a violation of the zone’s rules. Security zones are created by attaching a security zone recipe to a compartment. Security zone recipes are a collection of security zone policies that are applied to the security zone.



**Figure 1-5** Cloud Guard Dashboard

For example, a security zone policy such as deny block\_volume\_without\_vault\_key forces block volumes created in a security zone to require a customer-managed encryption key instead of the default Oracle-managed encryption key. These policies are not always intrusion oriented; for instance, the policy deny database\_without\_backup enforces all databases within that security zone to have automatic database backup configurations. Security policies are categorized into several groups, based on the security principles they help implement. You can create your own recipes using these policies to meet your specific security requirements. Apart from the custom recipes you can create, all tenancies come with a default recipe named Maximum Security Recipe that includes all available security zone policies.

When creating a new tenancy or starting with OCI, it is often good practice to enable Cloud Guard. Enabling Cloud Guard is an easy process that involves the tenancy owner granting the Cloud Guard service permissions to introspect resources within the tenancy. Although these services are part of OCI, they still require customers to acknowledge and provide privileges so that they can introspect customer-owned resources. Cloud Guard requires minimal permissions to monitor the resources in your tenancy. With Cloud Guard enabled, you can also use security zones, which requires an added set of permissions. Both Cloud Guard and security zones provide default configurations that you can customize to your needs or use as is when you get started.

### Service Limits and Cost Management

A basic understanding of service limits, quotas, budgets, and other guardrails and cost-management features in OCI can come in handy as you scale your workloads on OCI. Every tenancy is preconfigured at its creation with a set of limits for each resource type, called service limits. When you reach the service limit for a resource, OCI does not allow you to create more of that resource. These limits exist to protect users from accidentally creating too many resources. Service limits are updated automatically, based on consumption patterns, but they also can be updated by raising a request from the OCI web console.

Whereas service limits are set by Oracle, compartment quotas are limits that you can set on compartments. Compartment quotas are set through policies. Consider the following policy:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01_images.xhtml#f012-01a)

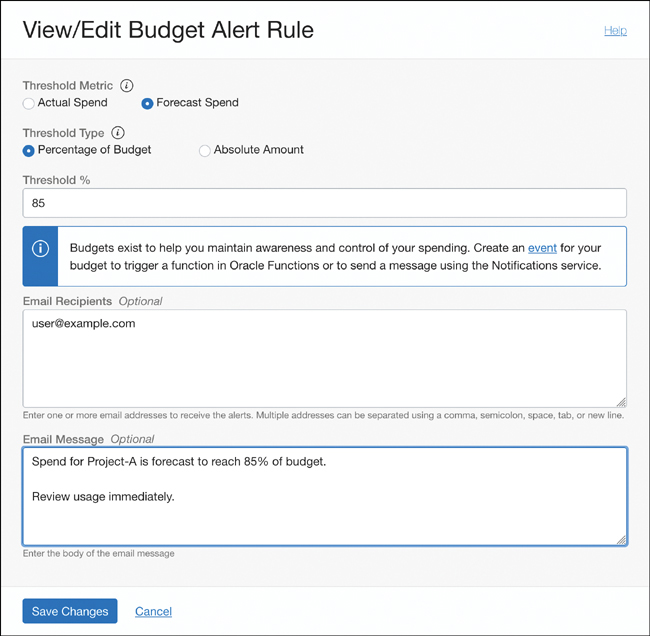
set compute-core quotas gpu-a10-v2-count to 20 in compartment Project-A

This policy caps the number of GPUs for shapes in the VM.GPU.A10 and BM.GPU.A10 series to 20 in the compartment named Project-A. Compartment quotas are typically used when a project or a team is given its own compartment and the admin wants to cap the resource usage for the project or team by setting a compartment quota. When the quota has been reached, no more resources of that kind will be provisioned in that compartment. In the preceding example, gpu-a10-v2-count is the quota for the number of GPUs that are available through compute shapes such as VM.GPU.A10 and BM.GPU.A10. These are GPU-enabled compute shapes; a tenancy admin might want to use quotas to ensure that all projects and teams have access to these resources so that no single team can monopolize its use or overconsume them.

These quota policies can be used to set a quota (set), unset a quota (unset, typically to override a tenancy-wide quota), or remove access to a resource within the specified compartment (zero). It is important to note that, when the compartment quota has been reached, the tenancy still has resources that can be consumed in other compartments. The tenancy administrator, or any user who has access to manage compartment quotas, is in full control of these limits. Each service exposes certain resource quotas; the OCI documentation provides a complete reference.[**3**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01.xhtml#ref1_3a)

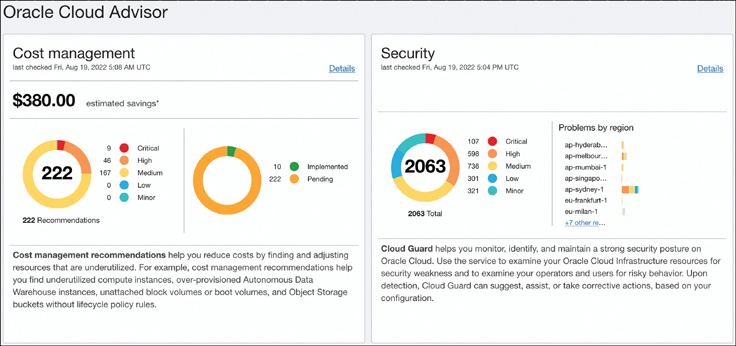
Limiting resource usage often can be a harsh step, especially when your goal is to minimize friction and increase team velocity while staying informed about resource usage and projections. To address this, OCI comes with features that help you track your cost across multiple dimensions. A budget is a construct that can be used to create your own soft limits on resources so that you can keep track of your infrastructure spending and create alerts to give you a heads-up. Alerts can be triggered when your spending crosses thresholds that you set. Administrators and users who have been given access to the feature can manage budgets using a consolidated view in OCI and also receive email alerts. Unlike compartment quotas, which are always set at a compartment level, budgets offer the flexibility of using cost-tracking tags or compartments to track spending. Budgets are not evaluated in real time and they update periodically (usually hourly). Each budget shows the current spend as well as a forecast for the spend, along with the last time the budget was evaluated. [Figure 1-6](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01.xhtml#ch01fig06) shows the alert options for a budget.

Because budgets are based on costs and spending instead of limits placed on resources themselves, they can be used alongside compartment quotas in a complementary manner. When bringing new workloads to OCI, it is typically easier to set a budget and work within that constraint. This is because, when bootstrapping a new workload, developers might not have a good handle on the resources required and the work typically involves a lot of initial flux in determining resource consumption, sizing, and setting up and tearing down ad-hoc environments. This is also the phase where you need the most velocity to build momentum for your team. You can keep tabs on the spending patterns using a budget. Then when you get a good sense of what your typical resource consumption is, you can set quotas to tighten the rules. This avoids accidental resource consumption, say, by a misbehaving automation.



**Figure 1-6** Creating a Budget Alert Rule

Apart from these services, OCI also includes many governance and compliance features. Tenancy explorer is a service that enables you to examine your tenancy to get a cross-region view of your resource usage. Cloud Advisor, shown in [Figure 1-7](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01.xhtml#ch01fig07), automatically and continuously analyzes your tenancy to find cost optimizations, performance improvements, and security posture improvements. Transparent billing offers analysis tools within the console that enable you to explore your spending in multiple dimensions.



**Figure 1-7** Oracle Cloud Advisor Dashboard

### Getting Started with Your Tenancy

Having covered the basic OCI terminology and concepts, this chapter moves on to look at how you can start setting up your tenancy to support development teams, create application environments, track costs, and set budgets. The first step is to sign up for an account. OCI offers a paid tier, a trial, and an always-free tier. The paid tier offers the full breadth of metered services, and you have multiple options to pay as you go and leverage existing licenses you might have. The trial (at press time) offers a 30-day window with a $300 credit to your account. The always-free tier limits the resources you can use, but these are available to you without charges for an unlimited time. After you have activated your account, you can log in through [https://cloud.oracle.com](https://cloud.oracle.com/) and see your cloud service dashboard.

#### Setting Up Users and Groups

When a tenancy is created, the email address you provided is added as the first user in the tenancy, and this user is added to the Administrators group. The Administrators group is a superuser group that, by default, has complete privileges over the entire tenancy. If you provided your email address when you signed up for your tenancy, then you are automatically an administrator once the tenancy has been created. Administrators can add other users and groups. The identity system can also be federated to an external identity provider.

Users can always change or reset their own console password, as well as manage their own API keys. An administrator does not need to create a policy to give a user those abilities.

To manage credentials for users other than yourself, you must be in the Administrators group or some other group that has permission to work with the tenancy. Having permission to work with a compartment within the tenancy is not sufficient. For more information, see the Administrators Group and Policy.[**4**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01.xhtml#ref1_4a)

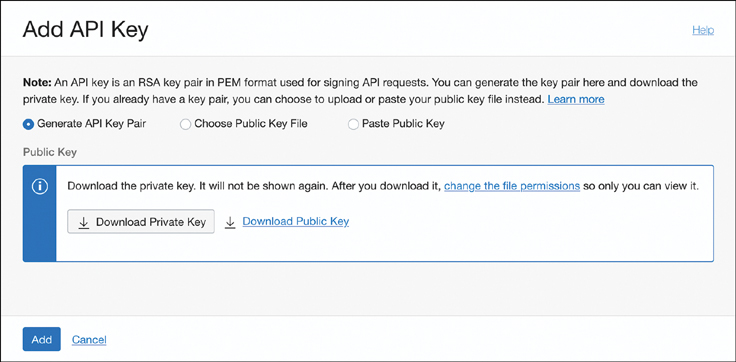
IAM administrators (or anyone with administration privileges to the tenancy) can use either the console or the API to manage all aspects of both types of credentials, for themselves and all other users. This includes creating an initial one-time password for a new user, resetting a password, uploading API keys, and deleting API keys.

#### Setting Up API Keys and Auth Tokens

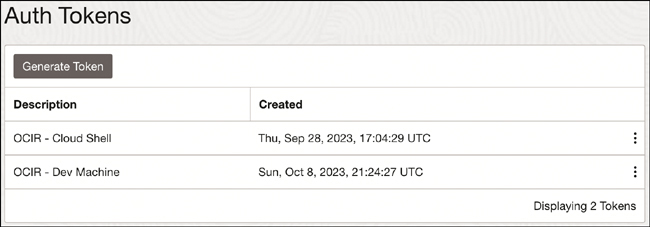
As a Developer or a DevOps engineer, one of the first actions to take after you have logged into your account, is to set up your API keys. The API key is a standard 2048-bit RSA key pair in PEM format. You upload the public key to your user profile and keep the private key securely. Every API call you make to OCI is signed by the private key, and OCI verifies the signature using the public key associated with your profile. The API keys securely authenticate you with the OCI APIs and are essential when you work with OCI tooling such as the command-line interface, Terraform, or the various SDKs. You can use the console to generate the private/public key pair for you (see [Figure 1-8](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01.xhtml#ch01fig08)). If you already have a key pair, you can choose to upload the public key as well. OCI tooling such as the OCI CLI or external tools such as Terraform use these API keys to sign the requests to OCI APIs and identify the user making the API calls. Because developers and DevOps engineers commonly work with multiple tenancies, regions or identities, these API keys and associated information are typically stored in a configuration file as profiles to easily switch between the configurations and identities. When you use the console to add the key pair, the console also generates a configuration file snippet that you can save. The configuration file typically has the following elements:

* user**:** The OCID that represents the user who owns this key
* fingerprint**:** The public key’s digest in two-digit groups
* tenancy**:** The tenancy’s OCID
* region**:** The default region to use
* key\_file**:** The path on your file system where the private key file is located

Auth Tokens are Oracle-generated token strings that can be used when a third party or API does not support key-based authentication and when OCI signature-based authentication cannot be used. Consider the example of using Docker to pull a container image from an image repository with the docker pull <image> command. OCI provides a private image repository that is Docker compatible. Docker, however, uses usernames and passwords to authenticate with repositories, not the more secure key-based authentication that OCI prefers. In this scenario, you can use an auth token. Auth tokens do not expire, and each user can have up to two auth tokens at any given time. Similar to API keys, users have the capability to create, update, and delete their own auth tokens. Administrators, on the other hand, have the capability to manage auth tokens for other users as well. [Figure 1-9](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01.xhtml#ch01fig09) shows a user profile with two auth tokens. The purpose of these tokens is clear from their descriptions; however, the actual token itself can no longer be viewed. If the token is lost, the user can delete and re-create another token.



**Figure 1-8** Generating an API Key Pair from the Console



**Figure 1-9** Generating Auth Tokens

### Planning How Your Teams Will Use OCI

As you prepare to onboard your team to OCI, it is prudent to consider a plan for how you will manage your tenancy and set it up so that multiple teams can work within it efficiently. You should consider how your IT and development organizations are structured and create a synergistic strategy for managing your tenancy. This includes planning how your users are grouped and designing an efficient compartment structure that makes it conducive for developers and DevOps teams to collaborate effectively. Development teams typically want quick and friction-free access to resources, whereas DevOps and SecOps teams have resource management and security posture as their top priorities. Designing an efficient compartment structure and a set of policies that manage access to these compartments is critical to ensuring success for the various stakeholder teams.

On one end of the spectrum, you can put all your resources into the root compartment. This would make the setup process simple, but at the same time it makes fine-grained access control more difficult. Although this might be useful during evaluation and basic prototyping with OCI, its untenable for real-world use in an enterprise setting. The other extreme would be having too many compartments with resources that are used and managed by the same groups located across several compartments, resulting in very complex policies needed to manage the right level of access to the resources and compartments. Both these extremes are uncommon for real-world use.

One common strategy would be to set up compartments and policies that align with the workloads. For instance, imagine that your team is bootstrapping a new project codenamed Hydra. You could create a new compartment named Hydra and then create groups such as Hydra-Devs, Hydra-Ops, Hydra-Admins, and Hydra-ReadOnly. All resources pertaining to the Hydra project, from core services such as networks, to developer lifecycle services such as DevOps, to application runtimes such as Kubernetes clusters, are all located within the compartment. The users on the project are grouped into the various groups. Some users could be taking on multiple roles by being a part of multiple groups. This compartment and group structure makes it easy to create policies that apply to each group and give them varying levels of access to the resources in the compartment. As an example, consider the following policy statements:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch01_images.xhtml#f017-01a)

Allow group Hydra-Admins to manage all-resources in compartment Hydra

Allow group Hydra-Ops to manage cluster-family in compartment Hydra

Allow group Hydra-Ops to manage devops-family in compartment Hydra

Allow group Hydra-Devs to use cluster-family in compartment Hydra

Allow group Hydra-Devs to use devops-family in compartment Hydra

Allow group Hydra-Devs to manage devops-repository in compartment Hydra

Allow group Hydra-ReadOnly to inspect all-resources in compartment Hydra

The example shows that Hydra-Admins can act as superusers within the compartment. This is often desirable to scale the team of tenancy admins while maintaining the principle of least access. A few users who belong to this group can assume the administrator role, but only within the confines of the Hydra project. The Hydra-Ops team has access to manage both the cluster-family as well as the devops-family. This means that members of this group can create and manage OKE clusters as well as code repositories (git), build pipelines, and more. The group Hydra-Devs can access these same resources but cannot make administrative changes such as creating or modifying the clusters; however, it can create code repositories. The Hydra-ReadOnly group can view all resources in the compartment but cannot otherwise affect them. This model makes it easy to create a sandboxed environment for each workload and enables administrators to create clear and concise policies, making them easy to understand and manage. This approach can be thought of as a vertical model because all layers of the stack for a particular workload are managed together.

Similar to organizing compartments and user groups around workloads, another common approach is to organize around the existing organizational structures. These cases might use compartments such as Networking, Databases, and Middleware, and user groups such as Network-Admins, DBAs, and DevOps that own and operate their own collection of resources for which they are responsible. In this model, the resources that are managed by a group are colocated in a compartment even when these are used by separate workloads. That is, regardless of which application they are associated with, all databases are in the same compartment and are administered by the same set of DBAs, which might feel familiar to organizations that follow a similar model in their enterprises. This model also makes it easier to manage policies for shared resources, such as databases that multiple applications use. Such an approach can be thought of as a horizontal model because it typically groups the management of a class of resources horizontally across workloads. This model can support organizational structures that already operate with these horizontal dependencies.

These are broad approaches that generalize common usage patterns. Each organization and each team’s dynamics are different, so it is a worthwhile exercise to plan how you want to organize your resources, group your users, and manage access. In addition to the approaches described here, the Oracle documentation covers other approaches and includes an OCI Cloud Adoption Framework that addresses best practices and provides concrete guidance across several pillars of success, such as business and people strategies, process design, technology implementation, and operations.

### Summary

This chapter just scratched the surface of OCI, but it gives you a sense of the scale at which you can operate; how the platform itself is structured; and what controls are available to manage your workloads, organize your teams, and provide access in a secure and efficient manner. This offers a starting point for you to explore more features in OCI and form preliminary ideas on how you can shape OCI to fit your deployment process and workflows. Elasticity and resiliency are keystones for a cloud native application. The next chapter discusses the infrastructure automation tools and platforms in OCI that enable you to scale infrastructure as your application’s needs change at runtime