Chapter 5
Building and Operationalizing Processing Infrastructure

Google Cloud Professional Data Engineer Exam objectives covered in this chapter include the following:

- **✓ 2.3 Building and operationalizing processing infrastructure. Considerations include:**
- Provisioning resources
- Monitoring pipelines
- Adjusting pipelines
- Testing and quality control>

Data engineering depends heavily on computing, or processing, resources. In this chapter, you will learn how to provision and adjust processing resources, including Compute Engine, Kubernetes Engine, Cloud Bigtable, and Cloud Dataproc. You will also learn about configuring managed, serverless processing resources, including those offered by App Engine, Cloud Functions, and Cloud Dataflow. The chapter also includes a discussion of how to use Stackdriver Metrics, Stackdriver Logging, and Stackdriver Trace to monitor processing infrastructure.

Provisioning and Adjusting Processing Resources

GCP has a variety of processing resources. For the purpose of the Professional Data Engineer exam, let's categorize them into server-based and serverless resources. *Server-based resources* are those that require you to specify virtual machines (VMs) or clusters; *serverless resources* do not require you to specify VMs or clusters but do still require some configuration.

The server-based services described here include the following:

- Compute Engine
- Kubernetes Engine
- Cloud Bigtable
- Cloud Dataproc
- Cloud Dataflow

Compute Engine can be configured using individual VMs or instance groups. The others are configured as clusters of VMs, sometimes with VMs taking on different roles within the cluster.

The serverless GCP services covered here include the following:

- App Engine
- Cloud Functions

Although you do not need to specify machine types or cluster configurations, you can specify some parameters to adjust these services to meet the requirements of different use cases.

PROVISIONING AND ADJUSTING COMPUTE ENGINE

Compute Engine supports provisioning single instances or groups of instances, known as *instance groups*. Instance groups are either managed or unmanaged. *Managed instance groups (MIGs)* consist of identically

configured VMs; unmanaged instance groups allow for heterogeneous VMs, but you should prefer managed instance groups unless you have a need for heterogeneous VMs. The configuration of a VM in a managed instance group is specified in a template, known as an *instance template*.

Provisioning Single VM Instances

The basic unit of provisioning in Compute Engine is a virtual machine. VMs are provisioned using the cloud console, the command-line SDK, or the REST API. Regardless of the method used to provision a VM, you can specify a wide range of parameters, including the following:

- Machine type, which specifies the number of vCPUs and the amount of memory
- Region and zone to create the VM
- Boot disk parameters
- Network configuration details
- Disk image
- Service account for the VM
- Metadata and tags

In addition, you can specify optional features, such as Shielded VMs for additional security or GPUs and tensor processing units (TPUs) for additional processing resources.

Compute Engine instances are created in the cloud console using a form like the one shown in Figure 5.1.

Instances can be provisioned from the command line using the `gcloud compute instances create` command. For example, consider the following command:

```
gcloud compute instances create instance-1 --zone=us-central1-a
               --machine-type=n1-standard-1 --subnet=default --network-
               --image=debian-9-stretch-v20191115 --image-project=debia
               --boot-disk-size=10GB --boot-disk-type=pd-standard
```

This command creates a VM of `n1-standard` type in the `us-centeral1-a` zone using the Premium network tier and the specified Debian operating system. The boot disk would be a 10 GB standard disk—in other words, not an SSD.

Once a VM instance is created, if the instance is stopped, you can adjust some configuration parameters. For example, you can detach and reattach a boot disk if it is in need of repair. You can also add or remove GPUs from stopped instances.

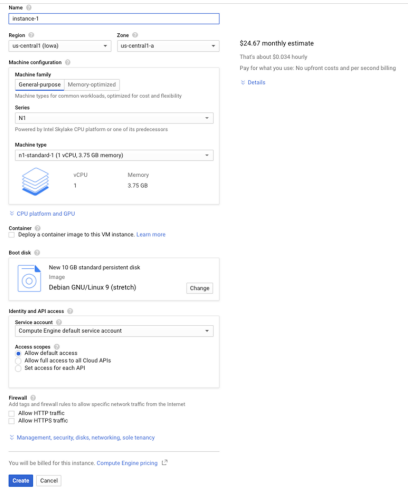

Figure 5.1 A basic VM instance provisioning form in the cloud console

Provisioning Managed Instance Groups

Managed instance groups are managed as a single logical resource. MIGs have several useful properties, including:

- Autohealing based on application-specific health checks, which replaces nonfunctioning instances
- Support for multizone groups that provide for availability in spite of zone-level failures
- Load balancing to distribute workload across all instances in the group
- Autoscaling, which adds or removes instances in the group to accommodate increases and decreases in workloads
- Automatic, incremental updates to reduce disruptions to workload processing

The specifications for a MIG are defined in an instance template, which is a file with VM specifications. A template can be created using the `gcloud compute instances-templates create` command. For example,

here is a command to create a template that specifies `n1-standard-4` machine types, the Debian 9 operating system, and a 250 GB boot disk:

```
gcloud compute instance-templates create pde-exam-template-custom \
    --machine-type n1-standard-4 \
    --image-family debian-9 \
    --image-project debian-cloud \
    --boot-disk-size 250GB
```

To create an instance from this template, you can use the `gcloud compute instances create` command seen earlier and include the `source-instance-template` parameter. For example, the following command will create an instance using the instance template just created:

```
gcloud compute instances create pde-exam-instance
            --source-instance-template=pde-exam-template-custom
```

You can also create an instance template using the cloud console. Figure 5.2 shows a typical form. Note that it is similar to the form used for creating an individual instance.

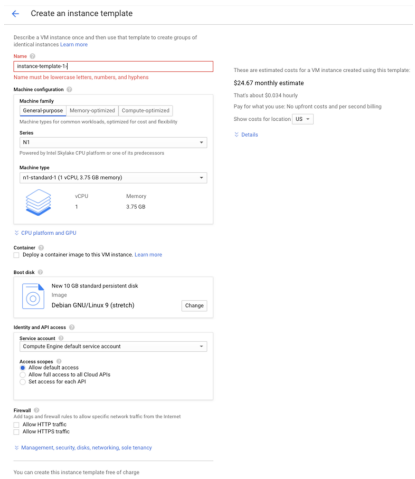

Figure 5.2 Form for creating an instance template

Once an instance template is created, you can create an instance group using the console as well, as shown in Figure 5.3.

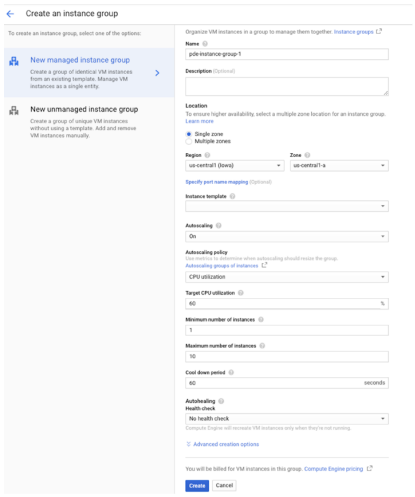

Figure 5.3 Form for creating a managed instance group

When creating an instance group, you can specify the name of the instance template along with a target CPU utilization that will trigger adding instances to the group up to the maximum number of instances allowed. You can also specify a minimum number of instances. You can also adjust the default cooldown period, which is the amount of time GCP will wait before collecting performance statistics from the instance. This gives the instance time to complete startup before its performance is considered for adjusting the instance group.

Adjusting Compute Engine Resources to Meet Demand

Creating and managing a single instance of a VM is an appropriate way to provision computing resources in some cases. For example, if you need a development instance or you want to run a small database for a local group of users, this is a reasonable approach. If, however, you want to be able to adapt quickly to changing workloads and provide high availability, a managed instance group is a better option.

One of the advantages of a managed instance group is that the number of VMs in the group can change according to workload. This type of horizontal scaling is readily implemented in MIGs because of autoscaling. An alternative, vertical scaling, requires moving services from one VM to another VM with more or fewer resources. Many data engineering use cases have workloads that can be distributed across VMs so that horizontal scaling with MIGs is a good option. If you have a high-performance computing (HPC) workload that must be run on a single server—for example, a monolithic simulation—then you may need to scale VMs vertically.

PROVISIONING AND ADJUSTING KUBERNETES ENGINE

Kubernetes Engine is a managed Kubernetes service that provides container orchestration. Containers are increasingly used to process workloads because they have less overhead than VMs and allow for finer-grained allocation of resources than VMs.

Some of this section originally appeared in Chapter 4, "Designing Compute Systems," of the *Official Google Cloud Certified Professional Cloud Architect Study Guide* (Wiley, 2019).

Overview of Kubernetes Architecture

A Kubernetes cluster has two types of instances: cluster masters and nodes. The *cluster master* runs four core services that control the cluster: controller manager, API server, scheduler, and etcd. The *controller manager* runs services that manage Kubernetes abstract components, such as deployments and replica sets. Applications running in Kubernetes use the API server to make calls to the master. The API server also handles inter-cluster interactions. The *scheduler* is responsible for determining where to run *pods*, which are the lowest-level schedulable unit in Kubernetes. The *etcd* service is a distributed key-value store used to store state information across a cluster.

Nodes are instances that execute workloads; they are implemented as Compute Engine VMs that are run within MIGs. They communicate with the cluster master through an agent called *kubelet*.

Kubernetes introduces abstractions that facilitate the management of containers, applications, and storage services. Some of the most important are as follows:

- Pods
- Services
- ReplicaSets
- Deployments
- PersistentVolumes
- StatefulSets
- Ingress

Pods are the smallest computation unit managed by Kubernetes. Pods contain one or more containers. Usually, pods have just one container, but if the services provided by two containers are tightly coupled, then they may be deployed in the same pod. For example, a pod may include a container running an extraction, transformation, and load process, as well as a container running ancillary services for decompressing and reformatting data. Multiple containers should be in the same pod only if they are functionally related and have similar scaling and lifecycle characteristics.

Pods are deployed to nodes by the scheduler. They are usually deployed in groups or replicas. This provides for high availability, which is especially needed with pods. Pods are ephemeral and may be terminated if they are not functioning properly. One of the advantages of Kubernetes is that it monitors the health of the pods and replaces them if they are not functioning properly. Since multiple replicas of pods are run, pods can be destroyed without completely disrupting a service. Pods also support scalability. As load increase or decreases, the number of pods deployed for an application can increase or decrease.

Since pods are ephemeral, other services that depend on them need a way to discover them when needed. Kubernetes uses the service abstraction for this. A *service* is an abstraction with a stable API endpoint and stable IP address. Applications that need to use a service communicate with the API endpoints. A service keeps track of its associated pods so that it can always route calls to a functioning pod.

A *ReplicaSet* is a controller that manages the number of pods running for a deployment. A *deployment* is a higher-level concept that manages ReplicaSets and provides declarative updates. Each pod in a deployment is created using the same template, which defines how to run a pod. The definition is called a *pod specification*.

Kubernetes deployments are configured with a desired number of pods. If the actual number of pods varies from the desired state—for example, if a pod is terminated for being unhealthy—then the ReplicaSet will add or remove pods until the desired state is reached.

Pods may need access to persistent storage, but since pods are ephemeral, it is a good idea to decouple pods that are responsible for computation

from persistent storage, which should continue to exist even after a pod terminates. *PersistentVolumes* is Kubernetes's way of representing storage allocated or provisioned for use by a pod. Pods acquire access to persistent volumes by creating a *PersistentVolumeClaim*, which is a logical way to link a pod to persistent storage.

Pods as described so far work well for stateless applications, but when state is managed in an application, pods are not functionally interchangeable. Kubernetes uses the StatefulSets abstraction, which is used to designate pods as stateful and assign a unique identifier to them. Kubernetes uses these to track which clients are using which pods and to keep them paired.

An *Ingress* is an object that controls external access to services running in a Kubernetes cluster. An Ingress Controller must be running in a cluster for an Ingress to function.

Provisioning a Kubernetes Engine Cluster

When you use Kubernetes Engine, you start by provisioning a cluster using either the cloud console, the command line, or the REST API. Figure 5.4 shows an example of the cloud console form for creating a Kubernetes cluster.

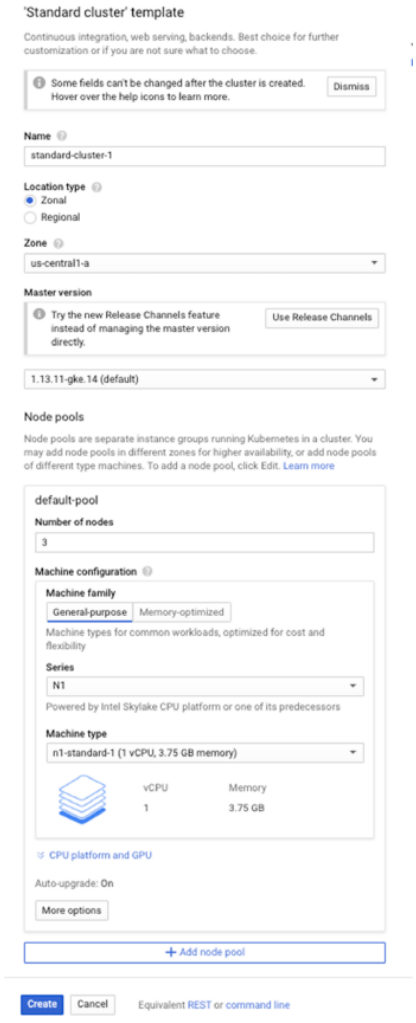

Figure 5.4 Cloud console user interface for creating a Kubernetes cluster

When creating a cluster, you will need to specify a cluster name, indicate whether the cluster will run in a single zone or in multiple zones within a region, and specify a version of GKE and a set of node pools.

To create a Kubernetes cluster from the command line, you can use the `gcloud container clusters create` command. For example, consider the following command:

```
gcloud container clusters create "standard-cluster-1" --zone "us-centra
                --cluster-version "1.13.11-gke.14" --machine-type "n1-
                --image-type "COS" --disk-type "pd-standard" --disk-si
                --num-nodes "5" --enable-autoupgrade --enable-autorepa
```

This command starts with `gcloud container clusters create`. Kubernetes Engine was originally called Container Engine, hence the use

of `gcloud container` instead of `gcloud kubernetes`. The zone, cluster versions, machine type, disk configuration, and number of nodes in the cluster are also specified. The `autoupgrade` and `autorepair` options are also specified.

Node pools are collections of VMs running in managed instance groups. Since node pools are implemented as MIGs, all machines in the same node pool have the same configuration. Clusters have a default node pool, but you can add custom node pools as well. This is useful when some loads have specialized needs, such as more memory or higher I/O performance. They are also useful if you want to have a pool of preemptible VMs to help keep costs down.

When jobs are submitted to Kubernetes, a scheduler determines which nodes are available to run the job. Jobs create pods and ensure that at least some of them successfully terminate. Once the specified number of pods successfully complete, the job is considered complete. Jobs include a specification of how much CPU and memory are required to run the job. If a node has sufficient available resources, the scheduler can run the job on that node. You can control where jobs are run using the Kubernetes abstractions known as *taints*. By assigning taints to node pools and tolerances for taints to pods, you can control when pods are run.

Adjusting Kubernetes Engine Resources to Meet Demand

In Kubernetes Engine, two important ways to adjust resources is by scaling applications running in clusters and scaling clusters themselves.

Autoscaling Applications in Kubernetes Engine

When you run an application in Kubernetes, you specify how many replicas of that application should run. When demand for a service increases, more replicas can be added, and as demand drops, replicas can be removed.

To adjust the number of replicas manually, you can use `kubectl`, which is the command-line utility for interacting with Kubernetes clusters. Note that this is not a `gcloud container` command; `kubectl` is used to control Kubernetes components. The `gcloud container` commands are used for interacting with Kubernetes Engine.

Let's take a look at an example `kubectl` command to adjust the number of replicas for a deployment named `pde-example-application`:

```
kubectl scale deployment pde-example-application --replicas 6
```

This command will set the number of desired replicas for the `pde-example-application` deployment to 6. There may be times when Kubernetes cannot provide all the desired replicas. To see how many replicas are actually running, you can use the `kubectl get deployments` command to list the name of deployments, the desired number of replicas, the current number of replicas, the number of replicas available to users, and the amount of time the application has been running in the cluster.

Alternatively, you can configure deployments to autoscale using the `kubectl autoscale` command. For example, consider the following:

```
kubectl autoscale deployment pde-example-application --min 2 --max 8 --
```

This command will autoscale the number of replicas of the `pde-example-application` between two and eight replicas depending on CPU utilization. In this case, if CPU utilization across all pods is greater than 65 percent, replicas will be added up to the maximum number specified.

Autoscaling Clusters in Kubernetes Engine

Kubernetes Engine provides the ability to autoscale the number of nodes in a node pool. (Remember, nodes are implemented as Compute Engine VMs, and node pools are implemented using managed instance groups.)

Cluster autoscaling is done at the node pool level. When you create a node pool, you can specify a minimum and maximum number of nodes in the pool. The autoscaler adjusts the number of nodes in the pool based on resource requests, not actual utilization. If pods are unscheduled because there is not a sufficient number of nodes in the node pool to run those pods, then more nodes will be added up to the maximum number of nodes specified for that node pool. If nodes are underutilized, the autoscaler will remove nodes from the node pool down to the minimum number of nodes in the pool.

By default, GKE assumes that all pods can be restarted on other nodes. If the application is not tolerant of brief disruptions, it is recommended that you not use cluster autoscaling for that application.

Also, if a cluster is running in multiple zones and the node pool contains multiple instance groups of the same instance type, the autoscaler tries to keep them balanced when scaling up.

You can specify autoscaling parameters when creating a cluster. Here is an example:

```
gcloud container clusters create pde-example-cluster \
               --zone us-central1-a \
               --node-locations us-central1-a,us-central1-b,us-central
               --num-nodes 2 --enable-autoscaling --min-nodes 1 --max-
```

This command creates a cluster called `pde-example-cluster` in the `us-central1-a` zone, with nodes in three zones and a starting set of two nodes per node pool. Autoscaling is enabled with a minimum of one node and a maximum of four nodes.

Kubernetes YAML Configurations

As with any other GCP service, we can use the command line, cloud console, and REST APIs to configure GKE resources. Kubernetes makes extensive use of declarative specifications using YAML files. These files contain information needed to configure a resource, such as a cluster or a deployment. Here, for example, is a deployment YAML specification for a deployment of an Nginx server using three replicas:

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
  name: "nginx-1"
  namespace: "default"
  labels:
    app: "nginx-1"
spec:
  replicas: 3
  selector:
    matchLabels:
      app: "nginx-1"
  template:
    metadata:
      labels:
        app: "nginx-1"
    spec:
      containers:
      - name: "nginx"
        image: "nginx:latest"
---
apiVersion: "autoscaling/v2beta1"
kind: "HorizontalPodAutoscaler"
metadata:
  name: "nginx-1-hpa"
  namespace: "default"
  labels:
    app: "nginx-1"
spec:
  scaleTargetRef:
    kind: "Deployment"
    name: "nginx-1"
    apiVersion: "apps/v1"
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: "Resource"
    resource:
      name: "cpu"
      targetAverageUtilization: 80
```

One of the advantages of using a managed Kubernetes service like GKE is that configuration files like this are generated for you when using command-line and cloud console commands.

PROVISIONING AND ADJUSTING CLOUD BIGTABLE

Cloud Bigtable is a managed wide-column NoSQL database used for applications that require high-volume, low-latency random reads and writes, such as IoT applications, and for analytic use cases, such as storing large volumes of data used to train machine learning models. Bigtable has an HBase interface, so it is also a good alternative to using Hadoop HBase on a Hadoop cluster.

Provisioning Bigtable Instances

Bigtable instance can be provisioned using the cloud console, command-line SDK, and REST API. When creating an instance, you provide an instance name, an instance ID, an instance type, a storage type, and cluster specifications. Figure 5.5 shows an example cloud console form for creating a Bigtable instance.

The instance type can be either production or development. Production instances have clusters with a minimum of three nodes; development instances have a single node and do not provide for high availability.

Storage types are SSD and HDD. SSDs are used when low-latency I/O is a priority. Instances using SSDs can support faster reads and are recommended for real-time applications. HDD instances have higher read latency but are less expensive and performant enough for scanning, making them a good choice for batch analytical use cases. HDD-configured instances cost less than SSD instances.

Clusters are sets of nodes in a specific region and zone. You specify a name, the location, and the number of nodes when creating a cluster. You can create multiple, replicated clusters in a Bigtable instance.

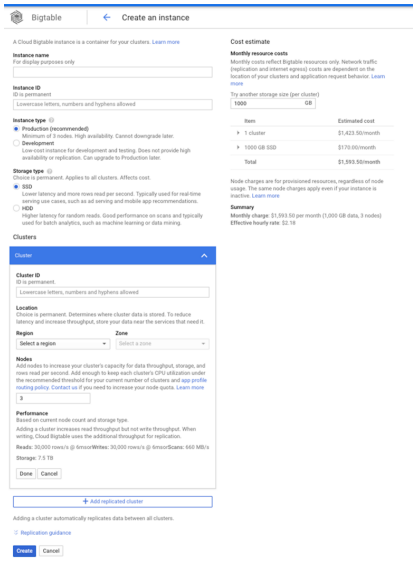

Figure 5.5 Form to create a Bigtable instance

Instances can be created using the `gcloud bigtable instances create` command. The following command creates an instance called `pde-bt-instance1` with one six-node cluster called `pde-bt-cluster1` in the `us-west1-a` zone that uses SSDs for storage. The instance is a production instance.

```
gcloud bigtable instances create pde-bt-instance1
    --cluster=pde-bt-cluster1 \
    --cluster-zone=us-west1-a
    --display-name=pdc-bt-instance-1 \
    --cluster-num-nodes=6 \
    --cluster-storage-type=SSD \
    --instance-type=PRODUCTION
```

Bigtable has a command-line utility called `cbt`, which can also be used to create instances along with other operations on Bigtable instances. Here is an example of a `cbt` command to create a Bigtable instance like the one created with the `gcloud` command earlier:

```
cbt createinstance pde-bt-instance1 pdc-bt-instance-1 pde-bt-cluster1 w
```

Once you have created a Bigtable instance, you can modify several things about the cluster, including:

- The number of nodes in each cluster
- The number of clusters
- Application profiles, which contain replication settings
- Labels, which are used to specify metadata attributes
- The display name

Replication in Bigtable

When multiple clusters are used, write operations are replicated across all clusters using a primary-primary configuration with eventual consistency. Bigtable supports up to four replicated clusters, and all clusters must be in their own zones. It is important to understand that running clusters in different regions will increase replication latency; however, this is still a reasonable choice for high-availability use cases that cannot tolerate a region-level failure.

Multiple clusters can help improve read performance as well. When running large batch analytic workloads along with write-intensive operations, users can experience degraded write performance. With multiple clusters, read operations can be routed to one cluster and write traffic to another cluster. This will separate write-intensive operations from batch read operations.

Bigtable allows some user configuration with respect to consistency guarantees. By default, Bigtable provides *eventual consistency*, which means that the data in clusters may not be the same sometimes, but they eventually will have the same data. If you need to ensure that your applications will not read data older than their latest writes, you can specify *read-your-writes consistency*. When all users must have a consistent view of data, you can specify *strong consistency*. That configuration will route all traffic to the same cluster, which will be configured with read-your-writes consistency. This configuration uses the other cluster only for failover.

PROVISIONING AND ADJUSTING CLOUD DATAPROC

Cloud Dataproc is a managed Hadoop and Spark service. When provisioning Cloud Dataproc resources, you will specify the configuration of a

cluster using the cloud console, the command-line SDK, or the REST API. When you create a cluster, you will specify a name, a region, a zone, a cluster mode, machine types, and an autoscaling policy. Figure 5.6 shows an example cloud console for creating a cluster.

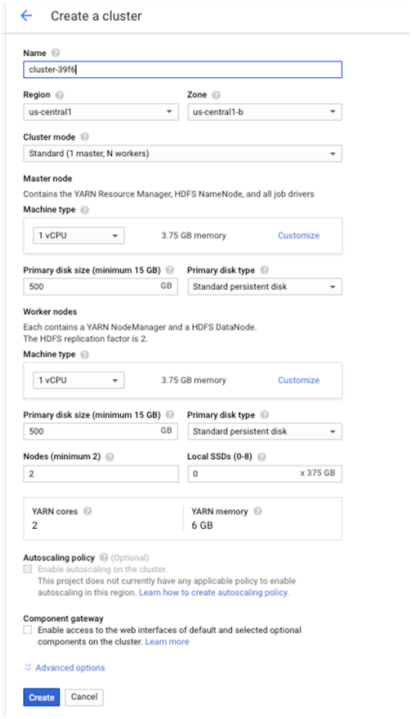

Figure 5.6 A cloud console form for creating a Cloud Dataproc cluster

The cluster mode determines the number of master nodes and possible worker nodes. The standard mode has one master and some number of workers. The single mode has one master and no workers. The high availability mode has three master nodes and some number of worker nodes. Master nodes and worker nodes are configured separately. For each type of node, you can specify a machine type, disk size, and disk type. For worker nodes, you can specify machine type, disk size and type, a minimum number of nodes, and optional local SSDs.

The `gcloud dataproc clusters create` command is used to create a cluster from the command line. Here is an example:

```
gcloud dataproc clusters create pde-cluster-1 \
               --region us-central1 \
               --zone us-central1-b \
               --master-machine-type n1-standard-1 \
               --master-boot-disk-size 500 \
               --num-workers 4
               --worker-machine-type n1-standard-1
               --worker-boot-disk-size 500
```

After a cluster is created, you can adjust the number of worker nodes, including the number of preemptible worker nodes. The number of master nodes cannot be modified. The number of worker nodes can also be adjusted automatically by specifying an autoscaling policy, which specifies the maximum number of nodes and a scale-up rate and a scale-down rate.

Configuring Cloud Dataflow

Cloud Dataflow executes streaming and batch applications as an Apache Beam runner. You can specify pipeline options when you run a Cloud Dataflow program. Required parameters are as follows:

- Job name
- Project ID
- Runner, which is DataflowRunner for cloud execution
- Staging locations, which is a path to a Cloud Storage location for code packages
- Temporary location for temporary job files

  You can also specify the number of workers to use by default when executing a pipeline as well as a maximum number of workers to use in cases where the workload would benefit from additional workers.

  There is an option to specify disk size to use with Compute Engine worker instances. This may be important when running batch jobs that may require large amounts of space on the boot disk.

  Cloud Dataflow does not require you to specify machine types, but you can specify machine type and worker disk type if you want that level of control.

CONFIGURING MANAGED SERVERLESS PROCESSING SERVICES

Several processing services in GCP are serverless, so there is no need to provision instances or clusters. You can, however, configure some parameters in each of the services. For the purposes of the Cloud Professional Data Engineer exam, we will review configuring the following:

- App Engine
- Cloud Functions

Configuring App Engine

App Engine is a serverless platform-as-a-service (PaaS) that is organized around the concept of a service, which is an application that you run in the App Engine environment. You can configure your service as well as supporting services by specifying three files:

- `app.yaml`
- `cron.yaml`
- `dispatch.yaml`

There is an `app.yaml` file associated with each version of a service. Usually, you create a directory for each version of a service and keep the `app.yaml` file and other configuration files there.

The `app.yaml` file has three required parameters: `runtime`, `handlers`, and `threadsafe`. `runtime` specifies the runtime environment, such as Python 3.

Support for Python 2 by the Python community was officially deprecated on January 1, 2020.

`handlers` is a set of URL patterns that specify what code is run in response to invoking a URL.

`cron.yaml` is used to configure scheduled tasks for an application. Parameters include a schedule of when to run the task and a URL to be invoked when the task is run.

The `dispatch.yaml` file is a place for specifying routing rules to send incoming requests to a specific service based on the URL.

Configuring Cloud Functions

Cloud Functions is another serverless processing service. This service runs code in response to events in Google Cloud. In addition to specifying the code to run, you have the option of configuring several parameters, including:

- `memory`
- `timeout`
- `region`
- `max-instances`

The amount of memory allocated to a function ranges from 128 MB to 2 GB.

`timeout` is the maximum time that the function is allowed to run before completing. If the function does not finish by the end of the timeout period, it is terminated. The default is 1 minute, but it can be set to as high as 9 minutes.

`region` is the geographical region in which the function will execute.

`max-instances` is the number of Cloud Function instances that will exist at any one time.

Monitoring Processing Resources

Once processing resources are provisioned and applications are running, it is important to monitor both the resources and the application performance. GCP includes *Stackdriver*, a set of monitoring tools that includes infrastructure and application monitoring, logging, and distributed tracing.

STACKDRIVER MONITORING

Stackdriver Monitoring collects metrics on the performance of infrastructure resources and applications. Resources can be in GCP as well as the Amazon Web Services cloud.

Stackdriver Monitoring collects data on more than 1,000 metrics, which vary by service type. For example:

- App Engine has metrics about the utilization of all VMs serving an application, the number of current active connections, and the total number of reserved cores.
- Compute Engine Autoscalers have metrics related to capacity and current utilization.
- BigQuery has metrics about query counts, execution times, scanned bytes, and table count.

- Cloud Bigtable collects data on CPU load, disk load, node count, bytes used, and storage capacity.
- Cloud Functions collects data on active instances, execution count, and execution times.

As you can see from the list, metrics are captured information about all the important characteristics of a service.

The data collected by Stackdriver Monitoring can be viewed on Stackdriver Monitoring dashboards and used for alerting.

Alerting allows you to specify a condition using metrics to identify when a resource is unhealthy or otherwise needs intervention. In Stackdriver Monitoring, alerts are created using an alerting policy. Alerting polices include conditions, such as CPU utilization exceeding 70 percent for three minutes, a notification channel, and optional documentation for those responding to the alert.

STACKDRIVER LOGGING

Stackdriver Logging is a service used for storing and searching log data about events in infrastructure and applications. Stackdriver Logging is able to collect data from more than 150 common applications.

Stackdriver Monitoring collects metrics, whereas Stackdriver Logging collects log entries. A *log entry* is a record about some event. These records may be created by an application, an operating system, middleware, or other service. A *log* is a collection of log entries.

Logs are maintained in Stackdriver for a specific period of time known as the *retention period*. If you want to keep them longer, you will need to export the logs before the end of the retention period. The retention period varies by log type. Admin activity audit logs, system event audit logs, and access transparency logs are kept for 400 days. Data access audit logs and other logs not related to auditing are kept 30 days.

Logs can be searched using a query language that supports pattern matching and Boolean expressions. You can use the query language to create filters that can be run from the Stackdriver Logging interface, as shown in Figure 5.7.

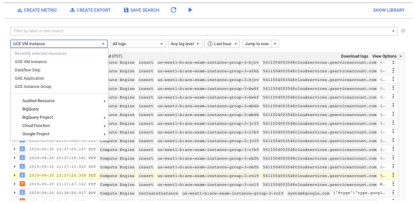

Figure 5.7 A example log listing in Stackdriver Logging

The query language can be used to specify log entries to export out of Stackdriver Logging. When you export log entries, they are written to a destination or sink, which could be Cloud Storage, BigQuery, or Cloud Pub/Sub. When a new log entry is written to a log, it is compared to export filters, and if there is a match, a copy of the log entry is written to the sink. Note that since only new log entries are written to the sink, this process cannot export log entries that already exist.

STACKDRIVER TRACE

Stackdriver Trace is a distributed tracing system designed to collect data on how long it takes to process requests to services. This is especially useful when you're using microservices architectures. Stackdriver Trace is available in Compute Engine, Kubernetes Engine, and App Engine.

Stackdriver Trace has two main components: a tracing client and a tracing user interface. The tracing client collects data and sends it to GCP. The tracing interface is used to view and analyze that trace data. The interface provides a curated experience that includes

- Performance insights
- Recent traces and times
- Most frequent URIs
- Daily analysis reports

Stackdriver Trace also provides a trace list interface that allows you to view traces in detail.

Exam Essentials

Know that Compute Engine supports provisioning single instances or groups of instances, known as instance groups. Instance groups are either managed or unmanaged instance groups. Managed instance groups (MIGs) consist of identically configured VMs; unmanaged instance groups allow for heterogeneous VMs, but they should be used only when migrating legacy clusters from on-premises data centers.

Understand the benefits of MIGs. These benefits include the following:

- Autohealing based on application-specific health checks, which replace nonfunctioning instances

- Support for multizone groups that provide for availability in spite of zone-level failures
- Load balancing to distribute workload across all instances in the group
- Autoscaling, which adds or removes instances in the group to accommodate increases and decreases in workloads
- Automatic, incremental updates to reduce disruptions to workload processing

Know that Kubernetes Engine is a managed Kubernetes service that provides container orchestration. Containers are increasingly used to process workloads because they have less overhead than VMs and allow for finer-grained allocation of resources than VMs. A Kubernetes cluster has two types of instances: cluster masters and nodes.

Understand Kubernetes abstractions. Pods are the smallest computation unit managed by Kubernetes. Pods contain one or more containers. A ReplicaSet is a controller that manages the number of pods running for a deployment. A deployment is a higher-level concept that manages ReplicaSets and provides declarative updates. PersistentVolumes is Kubernetes' way of representing storage allocated or provisioned for use by a pod. Pods acquire access to persistent volumes by creating a PersistentVolumeClaim, which is a logical way to link a pod to persistent storage. StatefulSets are used to designate pods as stateful and assign a unique identifier to them. Kubernetes uses them to track which clients are using which pods and to keep them paired. An Ingress is an object that controls external access to services running in a Kubernetes cluster.

Know how to provision Bigtable instances. Cloud Bigtable is a managed wide-column NoSQL database used for applications that require high-volume, low-latency writes. Bigtable has an HBase interface, so it is also a good alternative to using Hadoop HBase on a Hadoop cluster. Bigtable instances can be provisioned using the cloud console, the command-line SDK, and the REST API. When creating an instance, you provide an instance name, an instance ID, an instance type, a storage type, and cluster specifications.

Know how to provision Cloud Dataproc. When provisioning Cloud Dataproc resources, you will specify the configuration of a cluster using the cloud console, the command-line SDK, or the REST API. When you create a cluster, you will specify a name, a region, a zone, a cluster mode, machine types, and an autoscaling policy. The cluster mode determines the number of master nodes and possible worker nodes. Master nodes and worker nodes are configured separately. For each type of node, you can specify a machine type, disk size, and disk type.

Understand that serverless services do not require conventional infrastructure provisioning but can be configured. You can configure App Engine using the `app.yaml`, `cron.yaml`, `distpatch.yaml`, or `queue.yaml` file. Cloud Functions can be configured using parameters to specify memory, region, timeout, and max instances. Cloud Dataflow parameters include job name, project ID, running, staging location, and the default and maximum number of worker nodes.

Understand the purpose of Stackdriver Monitoring, Stackdriver Logging, and Stackdriver Trace. Stackdriver Metrics collect metrics on the performance of infrastructure resources and applications. Stackdriver Logging is a service for storing and searching log data about events in infrastructure and applications. Stackdriver Trace is a distributed tracing system designed to collect data on how long it takes to process requests to services.

Review Questions

You can find the answers in the appendix.

1. A group of data scientists wants to preprocess a large dataset that will be delivered in batches. The data will be written to Cloud Storage and processed by custom applications running on Compute Engine instances. They want to process the data as quickly as possible when it arrives and are willing to pay the cost of running up to 10 instances at a time. When a batch is finished, they'd like to reduce the number of instances to 1 until the next batch arrives. The batches do not arrive on a known schedule. How would you recommend that they provision Compute Engine instances?

   1. Use a Cloud Function to monitor Stackdriver metrics, add instances when CPU utilization peaks, and remove them when demand drops.
   2. Use a script running on one dedicated instance to monitor Stackdriver metrics, add instances when CPU utilization peaks, and remove them when demand drops.
   3. Use managed instance groups with a minimum of 1 instance and a maximum of 10.
   4. Use Cloud Dataproc with an autoscaling policy set to have a minimum of 1 instance and a maximum of 10.

2. You are running a high-performance computing application in a managed instance group. You notice that the throughput of one instance is significantly lower than that for other instances. The poorly performing instance

is terminated, and another instance is created to replace it. What feature of managed instance groups is at work here?

1. Autoscaling
2. Autohealing
3. Redundancy
4. Eventual consistency

3. A new engineer in your group asks for your help with creating a managed instance group. The engineer knows the configuration and the minimum and maximum number of instances in the MIG. What is the next thing the engineer should do to create the desired MIG?

1. Create each of the initial members of the instance group using `gcloud compute instance create` commands
2. Create each of the initial members of the instance group using the cloud console
3. Create an instance template using the `gcloud compute instance-templates create` command
4. Create an instance template using the `cbt create instance-template` command

4. Your team is migrating applications from running on bare-metal servers and virtual machines to running in containers. You would like to use Kubernetes Engine to run those containers. One member of the team is unfamiliar with Kubernetes and does not understand why they cannot find a command to deploy a container. How would you describe the reason why there is no deploy container command?

1. Kubernetes uses pods as the smallest deployable unit, and pods have usually one but possibly multiple containers that are deployed as a unit.
2. Kubernetes uses deployments as the smallest deployable unit, and pods have usually one but possibly multiple containers that are deployed as a unit.
3. Kubernetes uses replicas as the smallest deployable unit, and pods have usually one but possibly multiple containers that are deployed as a unit.
4. Kubernetes calls containers "pods," and the command to deploy is `kubectl deploy pod`.

5. A Kubernetes administrator wants to improve the performance of an application running in Kubernetes. They have determined that the four replicas currently running are not enough to meet demand and want to increase the total number of replicas to six. The name of the deployment is `my-app-123`. What command should they use?

1. `kubectl scale deployment my-app-123 --replicas 6`
2. `kubectl scale deployment my-app-123 --replicas 2`
3. `gcloud containers scale deployment my-app-123 --replicas 6`
4. `gcloud containers scale deployment my-app-123 --replicas 2`

6. A Cloud Bigtable instance with one cluster is not performing as expected. The instance was created for analytics. Data is continuously streamed from thousands of sensors, and statistical analysis programs run continually in a batch. What would you recommend to improve performance?

1. Use a write-optimized operating system on the nodes
2. Use a read-optimized operating system on the nodes
3. Add a cluster, run batch processing on one cluster, and have writes routed to the other cluster
4. Add another node pool to the cluster in each zone that already has a node pool or that cluster

7. A Cloud Dataproc cluster is running with a single master node. You have determined that the cluster needs to be highly available. How would you increase the number of master nodes to 3?

1. Use the gcloud dataproc clusters update command with parameter --num-masters 3
2. Use the gcloud dataproc clusters update command with parameter --add-masters 2
3. Use the cbt dataproc clusters update command with parameter --add-masters 2
4. The number of master nodes cannot be changed. A new cluster would have to be deployed with three master nodes.

8. You have provisioned a Kubernetes Engine cluster and deployed an application. The application load varies during the day, and you have configured autoscaling to add replicas when CPU utilization exceeds 60 percent. How is that CPU utilization calculated?

1. Based on all CPUs used by the deployment
2. Based on all CPUs in the cluster

3. Based on all CPU utilization of the most CPU-intensive pod
4. Based on the CPU utilization of the least CPU-intensive pod

9. A team of data scientists wants to run a Python application in a Docker container. They want to minimize operational overhead, so they decide to use App Engine. They want to run the application in a Python 3.4 environment. Which configuration file would they modify to specify that runtime?

1. app.yaml
2. queue.yaml
3. dispatch.yaml
4. cron.yaml

10. Your team is experimenting with Cloud Functions to build a pipeline to process images uploaded to Cloud Storage. During the development stage, you want to avoid sudden spikes in Cloud Functions use because of errors in other parts of the pipeline, particularly the test code that uploads test images to Cloud Storage. How would you reduce the risk of running large numbers of Cloud Functions at one time?

1. Use the --limit parameter when deploying the function
2. Use the --max-instances parameter when deploying the function
3. Set a label with the key max-instances and the value of the maximum number of instances
4. Set a language-specific parameter in the function to limit the number of instances

11. Audit control requirements at your company require that all logs be kept for at least 365 days. You prefer to keep logs and log entries in Stackdriver logging, but you understand that logs with predefined retention periods of less than 1 year will require you to set up an export to another storage system, such as Cloud Storage. Which of the following logs would you need to set up exports for to meet the audit requirement?

1. Admin activity audit logs
2. System event audit logs
3. Access transparency logs
4. Data access audit logs

12. You would like to collect data on the memory utilization of instances running in a particular managed instance group. What Stackdriver service would you use?

1. Stackdriver Debugger
2. Stackdriver Logging
3. Stackdriver Monitoring
4. Stackdriver Trace

13. You would like to view information recorded by an application about events prior to the application crashing. What Stackdriver service would you use?

1. Stackdriver Debugger
2. Stackdriver Logging
3. Stackdriver Monitoring
4. Stackdriver Trace

14. Customers are complaining of long waits while your e-commerce site processes orders. There are many microservices in your order processing system. You would like to view information about the time each microservice takes to run. What Stackdriver service would you use?

1. Stackdriver Debugger
2. Stackdriver Logging
3. Stackdriver Monitoring
4. Stackdriver Trace

15. You have created a test environment for a group of business analysts to run several Cloud Dataflow pipelines. You want to limit the processing resources any pipeline can consume. What execution parameter would you specify to limit processing resources?

1. numWorkers
2. maxNumWorkers
3. streaming
4. maxResources