Pre-Reading Material
Google Kubernetes Engine (GKE)

Before diving into the GKE course, it's essential to have a solid understanding of the following pre-requisite material. These foundational concepts will help you grasp the content more effectively and make the learning experience smoother.

Containers

A container is a unit of software that packages code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings.

Containers are similar to virtual machines (VMs) in that they both allow you to run software in an isolated environment. However, containers are much more lightweight than VMs. This is because containers do not need to include a complete operating system. Instead, they share the operating system of the host machine.

This makes containers much more efficient and portable than VMs. Containers can be easily moved from one host machine to another, and they can be scaled up or down more easily.

Here are some of the benefits of using containers:

- **Portability:** Containers can be run on any infrastructure that supports Docker. This makes it easy to deploy applications to different environments.
- **Efficiency:** Containers are lightweight and efficient, which can save on resources.
- **Scalability:** Containers can be easily scaled up or down to meet demand.
- **Isolation:** Containers are isolated from each other, which helps to prevent security vulnerabilities.
- **Reusability:** Containers can be reused for different applications.
- **Modularity:** Containers can be combined to create complex applications.

Workload deployment in a containerised environment

Workload deployment in a containerized environment is the process of deploying an application or service to a set of containers. This can be done manually or using a container orchestration tool.

When deploying a workload to a containerized environment, there are a few things to keep in mind:

- The application or service must be containerized. This means that it must be packaged in a container image that can be deployed to a container runtime.
- The container runtime must be available on the host machine. This is the software that runs the containers.
- The containers must be deployed to a network that can be accessed by the application or service.
- The containers must be configured to communicate with each other and with the outside world.

There are a few different ways to deploy a workload to a containerized environment:

- **Manual deployment:** This involves manually creating and deploying the containers. This is the least scalable and reliable method, but it is the simplest to get started with.
- **Container orchestration tool:** A container orchestration tool automates the deployment, scaling, and management of containers. This is the most scalable and reliable method, but it can be more complex to set up and use.

Using an orchestration tool along with proper monitoring setup can enhance the deployments of workloads to containers:

- Use a container orchestration tool: Container orchestration tools can help to automate the deployment, scaling, and management of containers, which can help to improve security and reduce the risk of errors.
- Use a monitoring tool: A monitoring tool can help to track the health of containers and ensure that they are running properly.

Stateful and Stateless Applications

Stateful and stateless applications are two different types of applications that are classified based on how they handle state.

- Stateful applications are applications that maintain state. This means that they remember information about their previous interactions with users or other applications. For example, a stateful web application might remember the items that a user has added to their shopping cart.
- Stateless applications do not maintain state. This means that they do not remember any information about their previous interactions. For example, a stateless web application might simply return the same HTML page to every user who requests it.

The main difference between stateful and stateless applications is how they handle state. Stateful applications store state in memory or on disk. This means that they can be restarted without losing their state. Stateless applications do not store state. This means that they must be restarted from scratch every time they are started.

Stateful applications are more complex to develop and deploy than stateless applications. This is because stateful applications need to be able to store and manage state. Stateless applications are simpler to develop and deploy, but they may not be able to provide the same level of functionality as stateful applications.

The best type of application to use depends on the specific requirements of the application. If the application needs to remember information about its previous interactions, then a stateful application is the best choice. If the application does not need to remember any information about its previous interactions, then a stateless application is the best choice.

Here are some examples of stateful applications:

- Online shopping carts
- Banking applications
- CRM applications
- Content management systems

Here are some examples of stateless applications:

- Web servers
- Load balancers
- Reverse proxies
- DNS servers

Autoscaling

Autoscaling is a cloud computing feature that dynamically adjusts the number of resources (such as CPU, memory, and storage) allocated to an application based on the demand. This helps to ensure that the application has the resources it needs to perform well, while also avoiding overprovisioning resources, which can waste money.

There are two main types of autoscaling:

- Vertical autoscaling changes the configuration of machines supporting the application by increasing or decreasing the number of resources allocated to virtual machine / host instance. For example, if an application is using a lot of CPU, autoscaling can be used to increase the number of CPU cores allocated to the instance running the application.
- Horizontal autoscaling increases or decreases the number of application instances. For example, if an application is experiencing a lot of traffic, autoscaling can be used to create more instances of the application.

Autoscaling can be used to improve the performance, reliability, and cost-efficiency of applications. By automatically adjusting the number of resources allocated to an application, autoscaling can ensure that the application has the resources it needs to perform well, even during periods of high traffic. Autoscaling can also help to avoid overprovisioning resources, which can waste money.

Here are some of the benefits of using autoscaling:

- **Improved performance**: Autoscaling can help to improve the performance of applications by ensuring that they have the resources they need to handle the load.
- **Improved reliability:** Autoscaling can help to improve the reliability of applications by ensuring that they are not overloaded during periods of high traffic.

- **Reduced costs:** Autoscaling can help to reduce costs by avoiding overprovisioning resources.

GCP Fundamentals

- **Projects** are the top-level containers for resources in GCP. Each project has its own set of resources, such as Compute Engine instances, Cloud Storage buckets, and Cloud SQL databases.
- **Resource hierarchy** is the way that resources are organized in GCP. Resources are organized in a tree-like structure to maintain the resources separated out based on their departments, business units, use cases or teams
- **IAM** (Identity and Access Management) is the system that GCP uses to control who has access to resources. IAM allows you to create users, groups, and roles, and then assign those roles to resources.
- **Networking** is the way that resources communicate with each other in GCP. GCP provides a variety of networking services, such as Cloud VPC, Cloud Load Balancing, and Cloud DNS.
- **Shared VPC** is a feature of GCP that allows you to share a single VPC network with multiple projects. This can be useful for organizations that want to centralize network administration.
- **Load Balancing** is a service that distributes traffic across multiple servers. This can help to improve the performance and reliability of applications.

Kubernetes Architecture

Kubernetes architecture is a system that automates deployment, scaling, and management of containerized applications. It is a popular choice for deploying microservices-based applications.

Kubernetes architecture consists of the following components:

- **Master node:** The master node is responsible for managing the cluster. It controls the worker nodes and ensures that the containers are running correctly.
- **Worker node:** The worker node is a physical or virtual machine that runs the containers for our applications.
- **Pods:** A pod is a group of one or more containers that are deployed together. Pods are the basic unit of deployment in Kubernetes.
- **Services:** A service is an abstraction that defines a set of pods that should be exposed as a single point of access. Services are used to make pods accessible to other pods or to external users.
- **Secrets:** Any sensitive information used by our application that you don't want to share with the outside world is known as secrets. This can be things like passwords.
- **Volumes:** Volumes are used to store data that is persisted across pod restarts. They can be used to store things like database data, logs, and other files.

Kubernetes Services

Kubernetes services are an abstraction that defines a set of pods that should be exposed as a single point of access. Services are used to make pods accessible to other pods or to external users making them scalable, available, and resilient.

A Kubernetes service has the following characteristics:

- It defines a set of pods that should be exposed as a single point of access.
- It can be exposed using a variety of protocols, such as HTTP, HTTPS, and TCP.
- It can be load balanced to distribute traffic across multiple pods.
- It can be configured to be resilient to failures.

There are different types of Kubernetes services:

- **ClusterIP:** This is the default type of service. It exposes the service on a private IP address within the Kubernetes cluster.
- **NodePort:** This type of service exposes the service on a specific port on each node in the cluster.
- **LoadBalancer:** This type of service uses an external load balancer to expose the service to the outside world.

Kubernetes Pods

A Kubernetes pod is a group of one or more containers that are deployed together and managed as a single unit. Pods are the smallest unit of deployment in Kubernetes.

A pod has the following characteristics:

- It is a group of one or more containers.
- The containers in a pod share the same network and filesystem.
- The containers in a pod are scheduled to run on the same node.
- Pods are ephemeral, meaning that they can be created and destroyed quickly.

Pods are used to deploy and manage containerized applications. They are a powerful tool for ensuring that applications are scalable, available, and resilient.

Kubernetes Networking

Kubernetes networking is the way that pods communicate with each other and with the outside world. Kubernetes provides a number of features for networking, including:

- **Pods:** Pods are the basic unit of networking in Kubernetes. Pods share the same network namespace, which means that they can communicate with each other by default.
- **Services:** Services are used to expose pods to other pods or to external users. Services can be exposed using a variety of protocols, such as HTTP, HTTPS, and TCP.
- **Network policies:** Network policies are used to control how pods can communicate with each other. Network policies can be used to restrict traffic between pods or to allow traffic only from certain pods.
- **Ingress:** Ingress is used to expose services to the outside world. Ingress can be used to route traffic to different services based on the URL path.

Kubernetes networking is a complex topic, but it is an important part of Kubernetes.

One of the key concepts related to Kubernetes networking isNetwork policies.

Network policies are used to control how pods can communicate with each other. Network policies can be used to restrict traffic between pods or to allow traffic only from certain pods.

Kubernetes Ingress and Egress

Ingress and Egress are two important concepts in Kubernetes networking.

- Ingress is a collection of rules that control how external traffic is routed to services within a Kubernetes cluster. Ingress rules can be used to route traffic to different services based on the URL path, the host name, or other criteria.
- Egress is a collection of rules that control how traffic from within a Kubernetes cluster is routed to the outside world. Egress rules can be used to restrict traffic to certain destinations or to allow traffic only to certain ports.

Here are some of the key concepts related to Kubernetes Ingress and Egress:

- **Ingress controller:** An Ingress controller is a software component that implements Ingress rules. Ingress controllers can be deployed as Kubernetes daemonsets or as standalone applications.
- **Ingress class:** An Ingress class is a way to group Ingress controllers together. Ingress classes can be used to select an Ingress controller based on its configuration or performance characteristics.
- **Ingress rule:** An Ingress rule is a rule that defines how traffic is routed to a service. Ingress rules can be defined using a variety of criteria, such as the URL path, the host name, or the protocol.

Basic familiarity of Unix CLI

Here are some of the basics of Unix CLI:

- **The command line:** The command line is a text-based interface that allows you to interact with the operating system. You can use the command line to run programs, manage files, and perform other tasks.
- **Commands:** Commands are instructions that you give to the operating system. There are many different commands available, each with its own purpose.
- **Arguments:** Arguments are additional information that you can provide to commands. Arguments can be used to specify the files that you want to work with, the options that you want to use, and other information.

- **Wildcards:** Wildcards are characters that can be used to match multiple files or directories. For example, the asterisk (*) wildcard can be used to match any number of characters.
- **Pipes:** Pipes are a way to connect the output of one command to the input of another command. This can be used to chain together commands to perform complex tasks.
- **Redirection:** Redirection is a way to redirect the output or input of a command to a file or device. This can be used to save the output of a command to a file or to send the input of a command from a file.
- **Environment variables:** Environment variables are variables that are used to store information about the environment. You can use environment variables to store things like your username, the current directory, and the path to your home directory.

Here are some of the basic commands that you should know:

- cd: The `cd` command is used to change directories. The basic syntax is `cd directory_name`, where `directory_name` is the name of the directory that you want to change to. For example, to change to the `/etc` directory, you would type `cd /etc`.
- ls: The `ls` command is used to list the contents of a directory. The basic syntax is `ls [options] directory_name`, where `options` are optional flags that can be used to control the output of the command. For example, to list all of the files in the current directory, you would type `ls`. To list all of the files in the current directory, including hidden files, you would type `ls -a`.
- pwd: The `pwd` command is used to print the current working directory. The basic syntax is `pwd`.
- mkdir: The `mkdir` command is used to create a new directory. The basic syntax is `mkdir directory_name`, where `directory_name` is the name of the directory that you want to create. For example, to create a new directory called `my_dir`, you would type `mkdir my_dir`.
- rmdir: The `rmdir` command is used to remove an empty directory. The basic syntax is `rmdir directory_name`, where `directory_name` is the name of the directory that you want to remove. For example, to remove the `my_dir` directory, you would type `rmdir my_dir`.
- touch: The `touch` command is used to create a new file or to change the timestamp of an existing file. The basic syntax is `touch file_name`, where `file_name` is the name of the file that you want to create or modify. For example, to create a new file called `my_file`, you would type `touch my_file`.

- cat: The `cat` command is used to display the contents of a file. The basic syntax is `cat file_name`, where `file_name` is the name of the file that you want to display. For example, to display the contents of the `my_file` file, you would type `cat my_file`.
- grep: The `grep` command is used to search for a pattern in a file. The basic syntax is `grep pattern file_name`, where `pattern` is the pattern that you want to search for and `file_name` is the name of the file that you want to search. For example, to search for the pattern `hello` in the `my_file` file, you would type `grep hello my_file`.
- man: The `man` command is used to display the manual page for a command. The basic syntax is `man command_name`, where `command_name` is the name of the command that you want to get help for. For example, to get help for the `ls` command, you would type `man ls`.

SRE Basics: SLIs, SLOs, SLAs

SLIs, SLOs, and SLAs are all terms used in the context of service level management (SLM). SLM is the practice of ensuring that services meet the agreed-upon expectations of their users.

- SLIs (Service Level Indicators) are metrics that measure the performance of a service. For example, an SLI for a web application might be the number of requests per second that the application can handle.
- SLOs (Service Level Objectives) are targets for SLIs. For example, an SLO for the web application might be that it should be able to handle 100 requests per second 99.9% of the time.
- SLAs (Service Level Agreements) are contracts between a service provider and its users that define the SLOs and the penalties that will be incurred if the SLOs are not met. For example, an SLA for the web application might state that the provider will refund 10% of the monthly subscription fee if the SLO of 99.9% uptime is not met.

SLIs, SLOs, and SLAs are all important for ensuring that services meet the expectations of their users. SLIs provide a way to measure the performance of a service, SLOs set targets for that performance, and SLAs define the consequences of not meeting those targets.

Here is an example of how SLIs, SLOs, and SLAs can be used to measure the performance of a web application:

- SLI: The number of requests per second that the application can handle.
- SLO: The application should be able to handle 100 requests per second 99.9% of the time.
- SLA: If the SLO is not met, the provider will refund 10% of the monthly subscription fee.