02 - Google Cloud Regions and Zones
-: In questo passaggio, capiamo perché abbiamo bisogno di regioni e zone. Cosa sono le regioni e le zone e perché ne abbiamo bisogno? Consideriamo uno scenario semplice. Immagina che la tua applicazione sia distribuita in un data center nella regione di Londra. Quali sarebbero le sfide con una simile architettura? La prima sfida è l'accesso lento agli utenti delle altre parti del mondo. Se hai utenti a Mumbai, New York o Sydney, otterrebbero un accesso lento. Questo è anche chiamato alta latenza. Numero due, cosa succede se il data center va in crash? La tua applicazione non funziona, quindi hai una bassa disponibilità. Ora, come possiamo migliorarlo ulteriormente? Quello che facciamo è aggiungere un altro data center a Londra. Quindi hai un nuovo data center e hai distribuito l'applicazione lì dentro. Quindi ora abbiamo due data center. Ora, quali sarebbero le sfide con questo tipo di architettura? Rimane la sfida uno. Gli utenti di altre parti del mondo avranno comunque un accesso lento. La nostra sfida due è risolta. Anche se uno dei data center si bloccasse, cosa accadrebbe? La mia applicazione verrebbe servita dall'altro data center. Quindi la tua applicazione è ancora disponibile dall'altro data center. Ora aggiungiamo una nuova sfida. Cosa succede se l'intera regione di Londra non è disponibile? Cosa succederebbe? La tua applicazione non funziona. Ora come lo miglioriamo? Quello che facciamo è creare la stessa architettura in due regioni. Quindi stiamo correndo in una nuova regione chiamata Mumbai. Quali sarebbero le sfide ora? Sfida uno, l'accesso lento agli utenti di altre parti del mondo è parzialmente risolto. L'accesso da Mumbai sarebbe veloce ma l'accesso da altre regioni del mondo, ad esempio New York o Sydney, rimarrà ancora lento e puoi risolvere questo problema aggiungendo distribuzioni per le tue applicazioni in altre regioni. La sfida due era già stata risolta. Anche se uno dei data center va in crash, possiamo servirlo dall'altro data center. Ora la sfida tre, cosa succede se l'intera regione di Londra non è disponibile? Cosa succederebbe adesso? La tua domanda verrà servita da Mumbai. Ora abbiamo capito il motivo per cui abbiamo bisogno di regioni e zone. Vorremmo distribuire la nostra applicazione in più regioni perché vorremmo alta disponibilità e bassa latenza per i nostri utenti. Ora pensa a questo. Desideri configurare data center in più regioni in tutto il mondo. È facile? La risposta, ovviamente, è no, ed è qui che i fornitori di servizi cloud ci aiutano. Vediamo come GCP ci aiuta a risolvere questo problema nel passaggio successivo.
-: Bentornato. Nella fase precedente abbiamo parlato del fatto che la creazione di data center in diverse regioni del mondo non è facile ed è qui che tutti i fornitori di servizi cloud, incluso Google, ci forniscono regioni in tutto il mondo. Google fornisce oltre 20 regioni in tutto il mondo, un elenco in continua espansione. Ci sono nuove regioni aggiunte ogni anno. Una regione non è altro che una posizione geografica specifica per ospitare le tue risorse. Quindi puoi decidere che vorrei ospitare le mie candidature nella regione di Mumbai o posso dire: "Vorrei ospitare le mie candidature nella regione di Londra o nella regione di Sydney". Avendo più regioni in tutto il mondo, Google Cloud semplifica molto, molto facilmente il deployment delle applicazioni in queste regioni. L'importante vantaggio di avere più regioni in tutto il mondo è l'elevata disponibilità. Se distribuisci la tua applicazione in più regioni in tutto il mondo, anche se una di queste regioni è inattiva, puoi servire l'applicazione dalle altre regioni. Bassa latenza. Puoi servire gli utenti dalla regione più vicina a loro e quindi ottengono una bassa latenza. Impronta globale. Una startup in India potrebbe essere in grado di distribuire facilmente l'applicazione in più parti del mondo e quindi può creare applicazioni globali. L'ultimo vantaggio è il rispetto delle normative governative. Paesi diversi avranno normative diverse. Ad esempio, supponiamo che gli Stati Uniti vogliano che i dati relativi a tutti i loro cittadini risiedano solo negli Stati Uniti. In questo tipo di situazioni posso creare una regione negli Stati Uniti e archiviare i dati relativi ai clienti statunitensi solo in quella specifica regione. Quindi le regioni ci aiutano a ottenere alta disponibilità , bassa latenza, presenza globale e ci aiutano a rispettare le normative governative. Qual è la necessità di zone? Come si ottiene un'elevata disponibilità nella stessa regione? Quindi vorrei distribuire la mia applicazione solo nella regione di Mumbai. Forse c'è un regolamento governativo a causa del quale vorrei che la mia domanda fosse solo nella regione di Mumbai ma anche con quella restrizione vorrei un'elevata disponibilità. Come posso ottenerlo? Ecco dove all'interno di ogni regione ci sono più zone. In Google Cloud, ogni regione ha almeno tre zone. Il vantaggio di avere più zone è una maggiore disponibilità e tolleranza di caduta all'interno della stessa regione. Posso distribuire la mia applicazione in ciascuna di queste zone in modo che anche se una di queste zone si arresta in modo anomalo, la mia applicazione può essere servita dall'altra zona. Nel linguaggio di Google, ognuna di queste zone ha uno o più cluster discreti. In parole povere, ognuna di queste zone ha almeno uno o più data center. Una cosa importante da ricordare è che queste zone sono collegati con collegamenti a bassa latenza. Quindi, anche se distribuisci un'applicazione nella zona uno e il database nella zona due, otterrai prestazioni davvero buone. Pertanto, la piattaforma Google Cloud fornisce più regioni in tutto il mondo e ciascuna di queste regioni ha almeno tre zone. Vediamo alcuni esempi per questo adesso. La regione di esempio è quella degli Stati Uniti occidentali. Questo è nel continente del Nord America in un luogo chiamato Dallas o Dallas, comunque sia pronunciato. E all'interno di questo, ci sono tre zone e le zone sono denominate US West 1A, West 1B e West 1C. Consideriamo un altro esempio. Europe North One è una regione. Questo è ad Hamina, in

Finlandia, e puoi vedere che anche quella regione ha tre zone. Nord 1A, Nord 1B e Nord 1C. Un altro esempio è Asia South One che è più vicino a me, che è Mumbai, India. E puoi vedere che ha anche tre zone di disponibilità Asia South 1A, Asia South 1B e Asia South 1C. In questo passaggio abbiamo introdotto il concetto di regioni e zone nella piattaforma Google Cloud. Ci vediamo al prossimo passaggio.

03 - Google Compute Engine for Professional Cloud Architect
Istruttore: Bentornato. Ogni volta che desideri distribuire applicazioni in locale, crei macchine virtuali. Come si creano macchine virtuali in Google Cloud? Il modo in cui puoi farlo è utilizzando Google Compute Engine, GCE. In questa sezione ti aiuteremo a configurare la tua prima macchina virtuale in Google Cloud. Capirai anche una serie di concetti relativi a motori di calcolo, tipi di macchine, immagini di macchine. Capirai cos'è un indirizzo IP interno, un indirizzo IP esterno e un indirizzo IP statico. Imparerai anche come automatizzare la configurazione delle applicazioni utilizzando script di avvio, modelli di istanza e anche creando un'immagine personalizzata. Alla fine di questa sezione imparerai anche come risolvere i problemi che incontri con Google Cloud Virtual Machines. Sei pronto per iniziare a giocare con Google Compute Engine? Ci vediamo al prossimo passaggio.
-: Bentornato. In questo passaggio, iniziamo con un servizio Google Cloud molto, molto importante chiamato Compute Engine. Ogni volta che desideri distribuire applicazioni, avrai bisogno di server. E quando desideri distribuire applicazioni nel cloud, avrai bisogno di server virtuali. Ed è quello che Compute Engine ti consente di eseguire il provisioning in Google Cloud. Iniziamo con Compute Engine in questo passaggio specifico. Nei data center aziendali, le applicazioni vengono distribuite su server fisici, dove distribuisci le applicazioni nel cloud. Nel cloud, affitteremmo server virtuali. E le macchine virtuali sono i server virtuali in GCP. E come si effettua il provisioning delle macchine virtuali? Il servizio che utilizzeresti è Google Compute Engine. Quindi Google Compute Engine è il servizio nella piattaforma Google Cloud per il provisioning e la gestione delle tue macchine virtuali. E quali sono le caratteristiche che offre? Ti aiuta a creare e gestire il ciclo di vita delle istanze di macchine virtuali. Puoi creare una macchina dell'istanza virtuale. Puoi effettivamente avviare, terminare, riavviare o terminare un'istanza di macchina virtuale. Puoi anche implementare il bilanciamento del carico e il ridimensionamento automatico per più istanze VM. Se la tua applicazione viene distribuita su più istanze VM, ti consigliamo di disporre di un bilanciamento del carico per distribuire il carico tra di loro. Inoltre, vorresti essere in grado di eseguire il ridimensionamento automatico. Vorresti essere in grado di aumentare il numero di istanze in base al carico. Se hai molti utenti che utilizzano l'applicazione, ti consigliamo di aumentare il numero di istanze. Altrimenti, ti consigliamo di ridurre il numero di istanze. Compute Engine ti consente anche di collegare l'archiviazione alle tue istanze di macchine virtuali. Vorresti eseguire il tuo sistema operativo su un disco rigido e collegarlo alla tua macchina virtuale. Compute Engine ti consente anche di gestire la connettività e la configurazione di rete per le tue istanze VM. Ti consigliamo di assegnare un indirizzo IP per la tua istanza VM e vorresti essere in grado di utilizzare l'indirizzo IP per parlare con la tua macchina virtuale o per inviare una richiesta alla tua macchina virtuale. Pertanto, Google Compute Engine ti consente di gestire il ciclo di vita delle tue istanze di macchine virtuali. Fornisce una serie di funzionalità aggiuntive come il collegamento dell'archiviazione, il bilanciamento del carico, il ridimensionamento automatico e la gestione della connettività di rete alle istanze VM. Ora, sono sicuro che stai diventando audace con tutta la teoria di cui stiamo parlando. Quello che faremo nei prossimi passaggi è configurare un set di istanze VM come server Web o server HTTP e vorremmo distribuire il carico tra di loro utilizzando i bilanciatori del carico. In questo rapido passaggio, ci è stato presentato cos'è un Compute Engine. Google Compute Engine è un servizio che ti consente di creare macchine virtuali in GCP e ti offre una serie di funzionalità. Iniziamo a giocare con Google Compute Engine iniziando il passaggio successivo.
-: Bentornato. Iniziamo a giocare con Google Compute Engine in questo passaggio specifico. Quello che vorremmo fare è creare alcune istanze VM e giocare con esse. Oltre a ciò, verificheremmo il ciclo di vita di queste istanze, in pratica proveremmo ad avviare, arrestare, riavviare e provare una varietà di operazioni che puoi eseguire con esse. Inoltre, useremmo SSH per connetterci alle istanze della macchina virtuale. Vorremmo accedere tramite SSH alle istanze della macchina virtuale e provare ad eseguire alcuni comandi. Iniziamo con la creazione di alcune istanze di macchine virtuali. E per poterlo fare, andiamo su Google Cloud, Web Console. Puoi andare oltre e digitare l'URL, cloud.google.com. E qui, puoi fare clic su Console che presenta qui. Quindi questa è la Web Console o è semplicemente chiamata Console. Questa è una specie di interfaccia web in cui puoi giocare con tutte le risorse che fornisci come parte della tua Google Cloud Platform. Se vieni qui per la prima volta e sei ancora nella versione di prova gratuita, dovresti vedere un messaggio come questo. Quello che ti consiglio di fare è cliccare su chiudi. E ora non dovresti più vedere quel messaggio. Quando accedi a Google Cloud Console, dovresti vedere la pagina del dashboard in cui potresti vedere le informazioni sul progetto, potrai vedere le diverse risorse che fanno parte di questo particolare progetto, e le diverse API e le richieste ad esse correlate. Non preoccupiamoci di tutte queste cose. Ciò che ci interessa davvero per ora è creare un Compute Engine. Quindi posso digitare, compute engine, qui. Quindi qui, puoi andare su Compute Engine. Se è la prima volta che utilizzi Compute Engine, il caricamento di questa pagina richiederà un po' di tempo. Potrebbero essere necessari circa cinque minuti prima che tu possa vedere il pulsante Crea qui. Quindi stiamo utilizzando il servizio Compute Engine e vorremmo creare un'istanza VM. Andiamo avanti e diciamo, crea. Ci sono molti dettagli che devi fornire per creare un'istanza VM. Nella schermata successiva, possiamo riempire i dettagli. Il

primo è il nome, quindi darò un nome come, la mia prima VM. Ogni volta che creiamo risorse in Google Cloud Platform, possiamo associarvi delle etichette. Le etichette sarebbero, diciamo, l'ambiente, per esempio. Quindi diciamo che questo è l'ambiente di sviluppo per me. Quindi, aggiungerò un'etichetta chiamata, environment dev. Puoi anche aggiungere etichette per unità di business. Quindi diciamo che l'unità di business a cui è correlata sono le vendite. Quindi puoi aggiungere etichette diverse che indicano ulteriori informazioni sulla VM. Vado avanti e dico salva qui. Quindi abbiamo aggiunto un paio di etichette, sviluppo ambientale e vendite aziendali. Successivamente, dobbiamo scegliere la regione e la zona. Ti consiglierei di prendere le impostazioni predefinite. L'impostazione predefinita che mi viene suggerita è US Central una regione e US Central 1A come zona. Dopo aver scelto la zona e la regione, ciò che devi scegliere è l'hardware su cui vorresti eseguire la tua macchina virtuale. Questa è chiamata la configurazione della macchina. Google Cloud Platform ti offre una serie di famiglie di macchine ottimizzate per diversi tipi di applicazioni. Vedresti uso generale, che sono tipi di macchine per carichi di lavoro comuni, ottimizzato per costi e flessibilità. Se passi all'ottimizzazione per computer, vedresti tipi di macchine ad alte prestazioni per applicazioni ad alta intensità di calcolo. Ci sono anche memoria ottimizzata. Quindi questi sono tipi di macchine di memoria di grandi dimensioni per carichi di lavoro ad alta intensità di memoria. Quindi, in base all'applicazione che eseguirai, puoi scegliere questa specifica famiglia di macchine che desideri utilizzare. Per ora, scegliamo l'uso generale e atteniamoci all'impostazione predefinita suggerita. E se vai più in basso, c'è un'altra scelta importante, quale sistema operativo vorresti sulla tua macchina virtuale. Il modo in cui lo scegliamo è scegliendo l'immagine. Se fai clic su Cambia, sarai in grado di vedere tutte le diverse immagini rese disponibili da Google Cloud. Quindi vedresti che ci sono immagini relative a diversi sistemi operativi forniti. Quindi puoi vedere Debian e Red Hat, Linux, puoi vedere Ubuntu, Windows Server e molte altre opzioni. Per ora, scegliamo l'impostazione predefinita, che è Debian, rimaniamo con quella e diciamo cancel. Se scorri un po' verso il basso, vedrai anche qualcosa chiamato configurazione del firewall. Come discusso in precedenza, vorremmo eseguire un server HTTP sulla macchina virtuale e vorremmo consentire il traffico HTTP in modo che qualcuno dall'esterno possa accedere all'applicazione in esecuzione sulla tua macchina virtuale. E questo è il motivo per cui vorresti consentire il traffico HTTP. Quindi, nel firewall, puoi fare clic su Consenti traffico HTTP. Quindi sostanzialmente quello che abbiamo fatto fino ad ora, è stato dare un nome alla macchina virtuale, abbiamo attaccato alcune etichette con essa e abbiamo scelto l'hardware predefinito che non è altro che la famiglia di macchine. E abbiamo scelto il software predefinito, che non è altro che l'immagine utilizzata per creare la tua VM. Oltre a ciò, abilitiamo il firewall per consentire il traffico HTTP. Ora, se scorri un po' verso destra, puoi vedere anche i costi coinvolti. Puoi vedere che eseguirlo per un mese mi costerebbe circa $ 24, ovvero circa $ 0,03 all'ora. E questo verrebbe prelevato dal mio credito gratuito. Una delle cose importanti che ti consiglierei di fare è giocarci e interrompere immediatamente l'istanza che crei. Il tuo credito gratuito è molto, molto prezioso e non sprecarlo per mantenere le istanze in esecuzione. Quindi andiamo avanti e creiamolo per ora e immediatamente non appena la nostra demo è terminata, andiamo avanti e fermiamo o terminiamo l'istanza. Quindi andiamo avanti e creiamo l'istanza adesso. Quindi ho iniziato a creare l'istanza, la creazione dell'istanza richiederebbe un po' di tempo. Ci vediamo nel passaggio successivo con ulteriori informazioni sulle macchine virtuali.

-: Bentornato. Nell'ultimo passaggio, abbiamo iniziato a creare la nostra macchina virtuale e mentre la creazione della macchina virtuale è in corso, discutiamo alcune cose importanti sulle macchine virtuali che abbiamo creato. Ci sono un paio di scelte importanti che abbiamo fatto quando abbiamo creato le nostre macchine virtuali utilizzando il motore di calcolo di Google. Il numero uno è qual è l'hardware? Numero due, qual è il sistema operativo e il software che vorresti sulla tua specifica macchina virtuale? Cominciamo con la prima decisione che abbiamo preso. Qual è l'hardware su cui vorremmo far girare la nostra macchina virtuale? Quando parliamo dell'hardware di una macchina virtuale ci sono due cose importanti che devi capire. Uno, è la famiglia di macchine e due, è il tipo di macchina. Sono presenti diverse famiglie di macchine per diversi tipi di carichi di lavoro. Lo scopo generale è quello consigliato per la maggior parte dei carichi di lavoro. Questo fornisce il miglior rapporto qualità prezzo. Quindi, se desideri eseguire un'applicazione Web o un semplice server delle applicazioni, o se desideri creare un database di piccole o medie dimensioni, o se desideri eseguire un ambiente di sviluppo in questo tipo di situazioni , puoi scegliere una famiglia generica. Se disponi di carichi di lavoro di memoria estremamente elevati, desideri molta RAM per eseguire le tue applicazioni. In quel tipo di situazioni puoi scegliere famiglie ottimizzate per la memoria. Gli esempi sono quando si desidera un database di grandi dimensioni in memoria o se si desidera eseguire analisi in memoria. L'altro tipo di casi d'uso sono i carichi di lavoro ad alta intensità di calcolo e per questo tipo di casi d'uso sceglieresti famiglie ottimizzate per il calcolo. Ad esempio, se hai un'applicazione di gioco che vorresti eseguire, hai bisogno di molta CPU e in quel tipo di situazioni andresti per famiglie ottimizzate per il calcolo. Quindi la prima scelta che stai facendo è basata sul tipo di applicazione che vorresti eseguire. Stai decidendo la famiglia di macchine. Una volta scelta la famiglia di macchine, dovrai scegliere il tipo di macchina. Quanta quantità esatta di memoria o disco della CPU desideri? Per ciascuna delle famiglie di macchine ci sono diversi tipi di macchine. In precedenza

abbiamo visto che E2 era una famiglia di macchine. E per E2, ci sono diversi tipi di macchina. E2, standard due, E2, standard quattro, E2, standard otto, 16 e 32. Prendiamone uno, E2 standard due. E2 qui rappresenta la famiglia del tipo di macchina della famiglia. Standard indica il tipo di carico di lavoro che desideri eseguire. E due qui indica il numero di CPU. Puoi vedere che lo standard E2 due ha due VCPU. Lo standard E2 quattro ha quattro CPU virtuali Lo standard E2 otto ne ha otto, e così via. L'altra cosa che puoi osservare qui è che il disco di memoria e le capacità di rete aumentano insieme alle CPU virtuali. Quindi, se scegli un tipo di macchina più grande, avrai a disposizione una maggiore quantità di memoria e anche lo sconto e anche le prestazioni di rete saranno migliori. Qui puoi vedere che la memoria aumenta all'aumentare del numero di VCPU, quindi questa è la prima scelta. Qual è l'hardware su cui vorresti eseguire la tua macchina virtuale? Sceglieremmo prima la famiglia del tipo di macchina e poi sceglieremmo questo specifico tipo di macchina. E la seconda domanda è qual è il sistema operativo e qual è il software che vorremmo su un'istanza di macchina virtuale? Questo è ciò che decidiamo scegliendo l'immagine. Esistono due tipi di immagini presenti in Google Cloud. Uno è l'immagine pubblica. In precedenza, abbiamo scelto Debian come immagine per creare la nostra macchina virtuale. Questo è un esempio di immagine pubblica. Queste immagini pubbliche sono gestite da Google, da community open source o da fornitori di terze parti. L'altro tipo di immagini sono immagini personalizzate. Queste sono immagini create da te, che sono personalizzate da te per i tuoi progetti specifici. I takeaway importanti fino a quando non scegli il tuo hardware decidendo il tipo di macchina. Scegli il tuo software decidendo la tua immagine. Ora torniamo alla nostra macchina virtuale e vediamo se è pronta. Quando vado alle istanze VM, posso vedere che la mia macchina virtuale è attiva e funzionante. Questo è figo. Puoi anche vedere che è stato creato in una zona specifica, US Central uno A, e puoi vedere che c'è un IP interno e un IP esterno, che è assegnato. L'IP esterno è quello che possiamo utilizzare per parlare con questa macchina virtuale da Internet. Puoi anche accedere tramite SSH alla macchina virtuale utilizzando questo pulsante, che è presente qui. A parte questo, se fai clic su questa cosa specifica puoi vedere le diverse operazioni che puoi eseguire. Nell'istanza della macchina virtuale, puoi vedere che puoi arrestare l'istanza della macchina virtuale o puoi sospenderla, sospenderla. La sospensione è fondamentalmente come una pausa, quindi puoi metterla in pausa e riprenderla, oppure puoi resettarla. Il ripristino non è altro che un riavvio , quindi l'istanza verrebbe arrestata e riavviata immediatamente. Puoi anche eliminare l'istanza. Per ora, quello che vorrei fare è entrare in SSH ed eseguire alcuni comandi. Quello che farei è fare clic su SSH. Quando faccio clic su SSH posso vedere una notifica che le finestre popup sono bloccate. SSH si aprirebbe effettivamente come una finestra popup e questo è il motivo per cui avrei bisogno di abilitare i popup. E una volta abilitato i popup, sarei in grado di accedere tramite SSH all'istanza VM. Diciamo connettiti. Se riscontri problemi con l'avvio della macchina virtuale, ti consiglio di provare anche con Google Chrome. Ora sono in grado di accedere tramite SSH alla macchina virtuale e qui posso eseguire alcuni comandi. Ma la dimensione del modulo è molto, molto piccola, quindi quello che posso fare è entrare qui e cambiare effettivamente. Quindi cambierò il tema del colore attuale in chiaro e cambierei effettivamente la dimensione della tecnologia in più grande. Freddo. Saresti in grado di vederlo molto più chiaramente, immagino. E ora, posso entrare qui e digitare un comando. Chi sono? Puoi vedere che il mio nome utente è stampato qui. Stampiamo l'attuale PWD funzionante direttamente a Sprint. Garantisce l'attuale funzionamento diretto. Elenchiamo tutti i file che sono presenti qui. LS, puoi vedere che non ci sono file presenti in questa directory specifica. In questo passaggio, abbiamo avviato la nostra macchina virtuale e siamo stati in grado di accedervi tramite SSH. Giochiamoci inoltre, in questo passaggio successivo.

-: Bentornato! Nell'ultimo passaggio, abbiamo avviato la nostra macchina virtuale e siamo stati in grado di eseguire alcuni comandi su di essa. In questo passaggio, vediamo come configurare un server http sulla nostra macchina virtuale Compute Engine. Vorrei installare del software su questa macchina specifica, quindi prima di fare qualsiasi cosa devo diventare effettivamente un utente root. Quindi, direi, "sudo su". Questo è il comando per diventare effettivamente un utente root. E una volta che lo faccio, quello che vorrei fare è aggiornare tutti i pacchetti che sono presenti qui. Apt è un gestore di pacchetti, che è installato di default su Debian e posso eseguire l'aggiornamento di Apt per inserire tutte le ultime modifiche. Quindi, facciamo un "aggiornamento Apt". Ora, una volta aggiornato l'indice del pacchetto, quello che vorrei fare è installare Apache. Apache è il server http che vorremmo eseguire sulla macchina virtuale. Il modo in cui possiamo farlo è dire "Apt install Apache2". Quindi, Apache2 è il nome del pacchetto. Quindi, vorremmo installare il pacchetto Apache2 usando il gestore di pacchetti, Apt. Premiamo invio. Quindi, puoi vedere che ci sta fornendo un po' di informazioni sul pacchetto Apache. Mostra cosa verrebbe installato, di quanta memoria avrebbe bisogno e ci chiede: "Vuoi continuare? E io direi di sì. Puoi vedere che sta scaricando il pacchetto e lo sta installando qui. Quindi, Apache2 verrebbe installato tra poco. Ci è voluto circa un minuto per installare il servizio Apache e al termine, il servizio Apache è ora installato. Ora posso tornare all'istanza della macchina virtuale. Una delle cose interessanti da sapere è che se vedi apparire un pannello come questo, puoi fare clic su Nascondi pannello informazioni. Quindi, mostra e nascondi. Pertanto, quando selezioni una macchina virtuale specifica, puoi

fare clic su Mostra pannello informazioni. Mostrerebbe informazioni su quella specifica macchina virtuale e, se desideri nascondere il pannello delle informazioni, puoi anche nasconderlo. Potrebbe essere utile mentre giochi con la console di molto di più. Abbiamo avviato il server Apache e vorremmo dare un'occhiata alla pagina dietro. Il modo in cui possiamo farlo è facendo clic sull'IP esterno. Quindi, puoi entrare qui e fare clic sul collegamento all'IP esterno e dovresti vedere la homepage di Apache2 Debian apparire. Qui, vedrai molte informazioni sulla configurazione di Apache e vedresti che la root dei documenti Debian predefinita è questa cartella specifica. Ora, vorremmo personalizzare questa pagina. Quindi, invece di questa pagina, vorremmo che la nostra pagina entrasse qui. Come possiamo farlo? È lì che vorremmo per personalizzare qualcosa presente in questa directory specifica. Quindi, copierò questo percorso di directory da qui. Quindi, "/var/www/html". Quindi, dirò "ls/var/www/html". Assicurati di non avere un punto o altro e puoi premere invio. E puoi vedere che c'è un file index.html che è presente qui, ed è index.html dove vorremmo inserire il nostro contenuto. Echo, e diciamo ciao mondo. Ops! E metterei una doppia citazione per concludere questo. Quindi, "Echo Hello World". Quindi, questo stamperebbe "Hello World" qui. Ora, quello che vorrei fare è effettivamente inviarlo a un file. Quindi, posso usare un carattere pipe. Quindi, non è altro che il simbolo del maggiore di. Quindi, puoi reindirizzare l'output in questo file. Quindi, /var/www/html/index.html. Quindi, qui è dove vorremmo scrivere "Hello World". E premiamo invio. Ora, entriamo qui e aggiorniamo la pagina. Freddo! Ora vedo "Hello World" qui. Quindi, va bene!` Quindi, siamo in grado di creare un semplice server Apache, servendo una semplice pagina Hello World. Ora, proviamo a renderlo un po' più complesso. Diciamo che vorrei effettivamente avere il mio nome host qui. Invece di dire semplicemente ciao mondo, vorrei aggiungere effettivamente un nome host per dire che questo è il nome host da dove proviene questa risposta. Come lo posso fare? Iniziamo con Eco. Proviamo a stamparlo. Il modo in cui possiamo stampare il nome host è dicendo, nome host in dollari tra parentesi. Quindi, "$(nome host)". Quindi, questo stamperebbe effettivamente il nome host. Il nome host che abbiamo dato in precedenza a questa particolare macchina virtuale è la mia prima VM. E, simile a quello, puoi effettivamente fare un "echo hostname_I", questo ti darebbe l'indirizzo IP. Quindi, quello che vorrei fare è aggiungere queste informazioni al mio index.html. Quindi, iniziamo con la stampa effettiva, "echo Hello World from $(nome host) e premiamo invio. Vediamo cosa sarebbe successo. Sì, sta stampando "Hello World dalla mia prima VM". Sembra fantastico. Dirò ciao da questo nome host specifico e vorrò anche stampare l'indirizzo IP. Quindi, dirò dollaro, tra parentesi metterei il nome dell'host. trattino I e premi invio. Questo sta stampando "Hello World dalla mia prima VM" e l'indirizzo IP. Se entri qui, vedresti che questo è l'indirizzo IP interno. Quindi, 10.128.0.2 è l'indirizzo IP interno. Questo è ciò che viene stampato qui. Quindi, sembra fantastico. Quindi, quello che vorrei fare è effettivamente prendere questo messaggio e inviarlo a index.html. Così io' "echo Hell World" da questa cosa specifica, e vorrei convogliare questo. Dove vuoi convogliare questo? Questo particolare file. Quindi, mettiamolo nello spazio e vediamo se riusciamo a vederlo qui. Fantastico, puoi vedere "Hello World dalla mia prima VM, 10.128.0.2". Nel passaggio abbiamo installato Apache sulla nostra macchina virtuale. Hai visto che è stato molto, molto facile. Tutto ciò di cui avevamo bisogno era accedere tramite SSH, installare il software di cui avevamo bisogno e potevamo facilmente personalizzare la pagina che avremmo voluto visualizzare dal nostro server web Apache. Ci sono molte altre cose che vorremmo sapere sulle macchine virtuali, per impararli tutti. Sono sicuro che ti stai divertendo molto e ci vediamo nel passaggio successivo.
-: Bentornato. Nell'ultimo passaggio, abbiamo visto un paio di indirizzi IP assegnati a una macchina virtuale. Abbiamo visto l'IP interno e l'IP esterno, che è assegnato a questa macchina virtuale. Comprendiamo qualcosa in più sull'IP interno e sull'IP esterno in questo passaggio specifico. Gli indirizzi IP esterni sono indirizzabili via Internet. Siamo stati in grado di chiamare questo specifico IP esterno dall'esterno, giusto? sono su internet. E sto contattando una macchina virtuale specifica, che è installata all'interno di Google Cloud Platform. Quindi questo è indirizzabile su Internet. Gli indirizzi IP interni sono interni a una rete aziendale. In questo esempio specifico, la macchina virtuale è installata all'interno della rete GCP. E l'indirizzo IP interno può essere utilizzato solo all'interno della rete Google Cloud. Quindi non sarai in grado di utilizzare questo 10.128.0.2 e inviargli una richiesta. Questo non è permesso. Ogni volta che parliamo di indirizzi IP esterni, non puoi avere due risorse con lo stesso indirizzo IP. L'IP esterno viene pubblicato su Internet. E quindi, non puoi avere due risorse con lo stesso IP pubblico. Tuttavia, due diverse reti aziendali possono avere risorse con lo stesso indirizzo IP interno. Poiché gli indirizzi IP interni non sono disponibili su Internet, sono solo interni a qualche rete, indirizzi IP interni simili possono essere utilizzati in due reti diverse. Ogni volta che crei un'istanza VM, per impostazione predefinita le viene assegnato almeno un indirizzo IP interno. Ogni volta che crei un'istanza VM, puoi scegliere se desideri creare un indirizzo IP esterno per essa. Una cosa importante da ricordare è il fatto che ogni volta che si arresta un'istanza VM, l'indirizzo IP esterno viene perso. Vediamo ora gli IP interni ed esterni in gioco. Passiamo alla nostra macchina virtuale che abbiamo creato in precedenza. E quello che farei in realtà è cercare di fermare tutto questo. Quindi andiamo avanti e diciamo basta. E dirò che anche la funzione popup si interrompe. Prima di fermarci, questa macchina virtuale aveva un IP interno e un IP esterno. Vediamo cosa accadrebbe quando si fermerà. Puoi anche annotare l'IP esterno. In questo momento è

34.123.6.112. Potresti avere un IP esterno diverso , ma assicurati di annotarlo. Aspettiamo che la VM venga arrestata. Proviamo ad aggiornare questo. Freddo. C'è voluto un po' di tempo. Circa un minuto o giù di lì. E dopo ciò, la mia macchina virtuale viene arrestata. E quando la macchina virtuale viene arrestata, puoi vedere che è ancora assegnato a un IP interno. Tuttavia, non vi è alcun IP esterno che gli viene assegnato. Ora, quello che farei è entrare qui e dire avvia o riprendi. E dirò di ricominciare anche qui. Vediamo cosa sarebbe successo. Freddo. Posso vedere che l'avvio della macchina virtuale ha esito positivo e viene assegnato un IP esterno diverso. In precedenza, questo era l'IP esterno che utilizzavo. 34.123.6 e 112, e se lo aggiorno effettivamente in questo momento, vedresti che otterrei una pagina non trovata o otterrei qualche errore perché l'IP esterno è ora cambiato. Puoi vedere che anche la mia connessione SSH non funziona. Dice che la connessione tramite Cloud Identity Aware Proxy non è riuscita. Quindi posso effettivamente andare avanti e dire vicino anche per quello SSH. Poiché l'IP esterno è cambiato, l'IP esterno precedente non è più utile. Ora, quello che farei è di nuovo SSH. Quindi puoi vedere che SSH si sta aprendo come una finestra popup in questo momento. E qui, vorrei eseguire il comando per avviare il server Apache. In precedenza abbiamo interrotto e riavviato l'istanza. Prima di farlo, facciamo un sudo su per diventare root e dopo posso eseguire un comando. Servizio Apache 2. Questo è il servizio che abbiamo installato in precedenza. E possiamo dire inizio. Quindi vorremmo avviare il servizio Apache 2. E una volta che questo servizio è attivo e funzionante, posso entrare e utilizzare l'IP esterno. E puoi vedere che sta dicendo Hello, World dalla mia prima VM, 10.128.0.2. In questa fase, abbiamo compreso i concetti di indirizzi IP interni ed esterni. Gli indirizzi IP esterni sono indirizzabili su Internet. Tuttavia, gli indirizzi IP interni sono interni solo a una specifica rete aziendale. Non puoi usarli da internet. Abbiamo anche visto che quando arrestiamo e avviamo un'istanza VM, l'indirizzo IP interno rimane lo stesso. Tuttavia, l'indirizzo IP esterno viene modificato quando riavviiamo un'istanza VM. Il fatto che un indirizzo IP esterno cambi quando riavvio una macchina virtuale può essere problematico. Come lo aggiustiamo? Vediamolo nel passaggio successivo.
Istruttore: Nell'ultimo passaggio, abbiamo parlato di indirizzi IP esterni e abbiamo visto il fatto che un indirizzo IP esterno cambia quando interrompo e avvio una macchina virtuale. Come posso ottenere un indirizzo IP esterno costante per un'istanza VM? Uno dei modi più semplici per farlo è assegnare un indirizzo IP statico alla VM. Poco dopo, esamineremo una soluzione migliore utilizzando i bilanciatori del carico. Ma per ora usiamo effettivamente un indirizzo IP statico. Vediamo come utilizzare un indirizzo IP statico in questo momento. Vado alla nostra piattaforma Google Cloud e qui puoi digitare indirizzi IP esterni . Quindi stiamo andando in indirizzi IP esterni, che fa parte di qualcosa chiamato VPC, Virtual Private Cloud. Ed è qui che puoi vedere gli indirizzi IP esterni assegnati alle nostre macchine virtuali. Quindi qui puoi vedere che c'è un indirizzo IP esterno che è già stato creato per la nostra specifica macchina virtuale. Quindi, ogni volta che crei una macchina virtuale con un indirizzo IP esterno, puoi venire qui e puoi vedere i dettagli dell'indirizzo IP esterno, che viene creato. Tuttavia, ciò che vogliamo non è solo un indirizzo IP esterno , ma ciò che vogliamo è un indirizzo IP esterno statico. E come possiamo riservarlo? Il modo in cui possiamo riservarlo è chiedendo l'indirizzo statico di riserva. Quindi andrei avanti e direi riserva indirizzo statico. Quindi l'ho cliccato. E qui, ora posso andare avanti e dire il mio primo indirizzo IP statico. Quando crei un indirizzo IP statico puoi scegliere il livello di servizio di rete. Quindi che tipo di prestazioni di rete vorresti? Ci sono due livelli che vengono offerti. Uno è premium e standard. L'impostazione predefinita è premium. Ci andrei. Puoi anche scegliere il tipo di formato dell'indirizzo IP che desideri utilizzare. Vorrei utilizzare un indirizzo IP in formato quattro o un formato sei. Vorrei andare con l'indirizzo IP quattro. E puoi anche scegliere se vuoi creare un indirizzo statico regionale o un indirizzo statico globale. Qui, vorremmo assegnare questo indirizzo statico a una macchina virtuale. Una macchina virtuale è una risorsa creata in una zona specifica. Quindi per noi sarebbe sufficiente un indirizzo statico regionale. Vorrei andare avanti e dire riserva. Ora questo creerebbe effettivamente un indirizzo statico e puoi vedere il tipo qui. Quindi, per impostazione predefinita, qualunque indirizzo IP esterno sia assegnato direttamente a una VM è temporaneo. Effimero significa che cambia. Quindi, quando riavvii l'istanza, cambia. Tuttavia, l'indirizzo che abbiamo appena creato è statico. Puoi vedere l'indirizzo statico qui, 34.123.61.12. Ora vogliamo assegnare alla nostra macchina virtuale. Come possiamo farlo? Se in realtà riduci leggermente lo zoom, intendo dire che il pulsante di modifica è ben nascosto da Google Cloud. Quindi vedresti un pulsante chiamato cambia qui. Quindi sull'indirizzo IP statico, se rimpicciolisci un po' dovresti vedere un pulsante chiamato cambia qui. Ed è quello su cui devi fare clic per modificare l'indirizzo IP. E tornerò allo zoom normale e qui posso effettivamente collegarlo con una VM. Direi la mia prima VM. E direi, va bene. Quindi vedresti che quell'indirizzo IP statico è ora assegnato all'istanza VM e quando aggiorni vedresti che l'indirizzo temporaneo viene rilasciato automaticamente. Quindi, quando assegno un indirizzo IP statico a un'istanza VM, l'indirizzo IP precedente che le era stato assegnato viene automaticamente rimosso. Quindi, se effettivamente aggiorno questo, questo è l'indirizzo IP precedente che è stato assegnato. Questo era l'indirizzo IP esterno, che era effimero. E puoi vedere che questo non funziona più. E ora posso effettivamente usare questo indirizzo IP statico. Quindi posso prendere l'indirizzo IP statico e posso usarlo. E la cosa

sorprendente di questo indirizzo IP statico è che se torno alle istanze, ad esempio, torniamo indietro, digitiamo le istanze VM. E torniamo alle nostre istanze VM del motore del computer. E qui, diciamo che vado qui e dico, basta. La sosta richiederebbe un po' di tempo. Aspettiamo che si completi. Proviamo ad aggiornare. No, questa fermata è ancora in corso. Ok, ci è voluto circa un minuto. Ora la sosta è completa. Quindi, anche se abbiamo arrestato l'istanza VM, l'indirizzo IP statico è ancora assegnato alla tua istanza VM. In questa fase abbiamo esaminato come ottenere un indirizzo IP esterno costante per un'istanza VM. La soluzione era utilizzare un indirizzo IP statico. Abbiamo visto come creare un indirizzo IP statico e come possiamo assegnarlo a un'istanza VM. Parliamo un po' di più degli indirizzi IP statici nel passaggio successivo.
-: Bentornato. In questo passaggio, diamo un'occhiata ad alcune cose che devi ricordare sugli indirizzi IP statici. Una cosa importante che devi ricordare sugli indirizzi IP statici è che puoi effettivamente passare da un'istanza VM a un'altra istanza VM nello stesso progetto. Quindi, se crei effettivamente una nuova istanza VM qui, puoi facilmente cambiare l'indirizzo IP statico da quello attuale a quello nuovo. La prossima cosa importante da ricordare è che l'IP statico rimane collegato anche se interrompiamo l'istanza. Come puoi vedere qui, abbiamo interrotto l'istanza specifica. Tuttavia, l'IP statico viene assegnato. È importante ricordare che ti viene addebitato un IP statico anche quando non lo usi. Quindi verrai fatturato quando lo usi e quando non lo usi. Una cosa sorprendente è che il conto per non usarlo è molto più del conto che paghi quando lo usi. Quindi se hai un IP statico e non lo stai utilizzando, la raccomandazione è di rilasciarlo immediatamente. Quindi assicurati di rilasciare esplicitamente un IP statico quando non lo stai utilizzando. Andiamo avanti e facciamolo adesso, e quello che farò è entrare ed eliminare questa specifica istanza. Quindi andrei avanti ed eliminerei l'istanza VM. Quando aggiorno le istanze VM, Non ho davvero un'istanza VM. Ora, se digito effettivamente indirizzi IP esterni ed entro in una rete VPC, puoi vedere che l'indirizzo statico è ancora disponibile. Quindi questo indirizzo IP statico in questo momento non è utilizzato da nessuno e questo è il tipo di scenario in cui ti verrebbe addebitato. E questo è il motivo per cui la migliore pratica quando si tratta di indirizzi IP statici è andare avanti e rilasciarli. Quindi selezionerò questo e vado avanti e dico Rilascia indirizzo IP statico. Quindi, ogni volta che non utilizzi un indirizzo IP statico, la migliore pratica è procedere e rilasciarlo, perché ti verrà addebitato se non utilizzi un indirizzo IP statico. Quindi assicurati di rilasciare effettivamente in modo esplicito un IP statico quando non lo stai utilizzando. In questo passaggio, abbiamo imparato qualcosa in più sull'indirizzo IP statico. Ci vediamo al prossimo passaggio.
-: Bentornato. Negli ultimi passaggi, abbiamo creato una macchina virtuale. Abbiamo installato un server HTTP Apache su di esso. Tuttavia, ci sono molti passaggi nella creazione di una VM e nella configurazione del server Apache HTTP su di essa. Come si riduce il numero di passaggi nella creazione di un'istanza VM e nella configurazione di un server HTTP? Nei prossimi passaggi, esploreremo alcune opzioni per farlo. Il primo è lo script di avvio. Vorremmo eseguire uno script all'avvio di una macchina VM per installare automaticamente il server Apache. L'opzione successiva è un modello di istanza. L'ultima opzione che esploreremmo è un'immagine personalizzata. Nel passaggio, iniziamo con la prima opzione che è uno script di avvio. Come possiamo utilizzare uno script di avvio per configurare un server Apache HTTP all'avvio di un'istanza VM? Il processo di installazione di qualsiasi software o installazione di patch del sistema operativo, quando viene avviata un'istanza VM, è chiamato bootstrap. Ogni volta che avvii una macchina virtuale, vuoi essere sicuro che sia sicura. Come puoi essere sicuro che sia sicuro? Vuoi assicurarti che vengano applicate anche tutte le patch del sistema operativo, quelle più recenti. E una delle opzioni per poterlo fare è usare uno script di avvio. Nella macchina virtuale è possibile configurare uno script di avvio per il bootstrap. Vediamo ora una semplice demo utilizzando questo script di avvio. Quindi torniamo alle nostre istanze VM. Il motore di calcolo delle istanze VM è dove ci stiamo dirigendo. E al momento non abbiamo istanze di macchine virtuali. Va bene. Andiamo avanti diciamo crea. E qui, lo chiamerò "La mia VM con script di avvio". E sceglierò le impostazioni predefinite per la maggior parte delle cose. Vorrei consentire il traffico HTTP. Quindi firewall, consenti il traffico e vorrei installare del software all'avvio della macchina VM. E per questo, avremmo bisogno di configurare lo script di avvio. E per poterlo fare faremo clic su questa ordinanza di rete del disco di sicurezza di gestione. Si apriranno alcune schede. Parleremo di queste schede un po 'più tardi. Per ora, vuoi concentrarti su questo script di avvio. Quindi se scorri poco in basso nella gestione vedresti Startup Script. Puoi scegliere di specificare uno script di avvio che verrà eseguito all'avvio o al riavvio dell'istanza. Gli script di avvio possono essere utilizzati per installare software e aggiornamenti e per garantire che i servizi siano in esecuzione all'interno della macchina virtuale. E come parte del nostro script di avvio, cosa vuoi fare? Vogliamo fare un aggiornamento dell'app e quindi vogliamo installare Apache 2, quindi vogliamo impostare una pagina di esempio. Lo script esatto da utilizzare è presente anche nei passaggi precedenti, quindi se vuoi, puoi effettivamente copiarlo da lì o dal PPT. Quindi andrò allo script di avvio e incollerò questo. Quindi qui nello script di avvio, vogliamo usare Bash e stiamo dicendo aggiornamento dell'app e stiamo dicendo app, "-y", installa Apache 2. Quando noi installa Apache 2 senza usare "-y", farebbe una domanda: "Vuoi andare avanti e installare Apache 2?" E non vogliamo che venga fatta una domanda. Ecco perché

aggiungiamo un'opzione "-y" e l'eco è lo stesso di quello che avevamo prima. Quindi basta, andiamo avanti e diciamo crea. Le due cose che abbiamo personalizzato sono la prima che abilitiamo l'accesso HTP nel firewall. Numero due, Le istanze sono state avviate. Quello che ho consigliato di fare è di dargli effettivamente un paio di minuti, quindi anche dopo che le istanze sono pronte, ci vorranno un paio di minuti prima che tutte le installazioni del software avvengano perché stiamo installando alcune cose all'avvio del virtuale macchina. Quindi dagli un paio di minuti e poi fai effettivamente clic sull'IP esterno. E poi vedresti qualcosa del genere: "Hello world! From my VP with Startup Script 10.12803", e questo è l'IP interno che corrisponde qui. Nel passaggio abbiamo esaminato come automatizzare l'installazione delle patch del sistema operativo o di qualsiasi software che desideri installare quando avvii un'istanza VM. Lo abbiamo fatto usando lo script di avvio. Per quanto riguarda l'esame, ricordare lo script di avvio è molto, molto importante.
-: Bentornato. Nell'ultimo passaggio, abbiamo giocato con lo script di avvio e abbiamo semplificato l'installazione di un server HTTP su una macchina virtuale. Tuttavia, diciamo che vorrei creare un'altra istanza. Cosa avrei bisogno di fare? Devo entrare, scegliere l'hardware, scegliere il software e quindi configurare lo script di avvio, quindi etichettare HTTP firewall, quindi andare avanti e creare l'istanza. Come posso evitarlo? Come posso creare le mie istanze ancora più facilmente? La domanda da porsi è perché dobbiamo specificare tutti i dettagli dell'istanza VM? Ad esempio, immagine, tipo di istanza, script di avvio e tutto il resto ogni volta che avvii un'istanza? Che ne dici di creare un modello di istanza? Quando crei un modello di istanza puoi definire il tipo di macchina, l'immagine, le etichette, lo script di avvio e molte altre proprietà relative alla tua macchina virtuale, e puoi utilizzare quel modello per avviare successivamente le tue macchine virtuali. Quindi questo viene utilizzato per creare istanze di macchine virtuali e poco dopo vedremo anche qualcosa chiamato gruppi di istanze. È possibile avviare un gruppo di macchine virtuali. Puoi utilizzare i modelli di istanza per avviare singole istanze VM o un gruppo di istanze VM. Un aspetto importante di un modello di istanza è che non può essere aggiornato. Dopo aver creato un modello di istanza, non puoi modificarlo. Quello che puoi fare è effettivamente creare una copia del modello esistente , modificarlo e crearne una nuova versione. Un'altra caratteristica presente con il modello di istanza è invece di specificare un'immagine specifica, puoi anche specificare la famiglia di immagini. Quando specifichi una famiglia di immagini, viene utilizzata l'ultima versione non obsoleta di quella specifica famiglia. Proviamo ora a creare un modello di istanza e creare un paio di istanze VM. Come posso creare un modello di istanza? Penso che sia molto facile, giusto? Posso andare al modello di istanza e crearlo. Ora, prima, quello che farei è Vorrei entrare nell'istanza che è già in esecuzione e copiare lo script di avvio per esso. Quindi, se scorri verso il basso e vai alla parte dei metadati e vai ai metadati personalizzati, è qui che puoi vedere questo script di avvio. Quello che vorrei fare è usare questo script di avvio, quindi lo copierò. Puoi anche copiarlo e impaginarlo in un file di testo in modo da poterlo riutilizzare più volte. Vorrei andare ora ai modelli di istanza e non vogliamo creare un modello di istanza , quindi crea un nuovo modello di istanza. Lo chiamerei il mio modello di istanza con lo script di avvio. Puoi vedere che non ho davvero bisogno di specificare una regione o una zona quando creo un modello di istanza perché il modello di istanza può essere utilizzato per creare risorse in regioni e zone diverse. Questo è il motivo per cui non abbiamo bisogno di configurare una regione o una zona. Sceglierò il tipo di macchina predefinito, che è consigliato. Sceglierò l'avvio predefinito che è consigliato e dirò, consenti il traffico HTTP, e andrò in gestione, sicurezza, dischi e rete. Questo sarebbe lo stesso di quello che avremmo quando configuriamo una macchina virtuale e qui andrei a configurare uno script di avvio. Quindi questa è la stessa cosa che abbiamo configurato in precedenza, e andiamo avanti e diciamo Crea. con la creazione di un modello di istanza. Tuttavia, quando creiamo istanze utilizzando il modello di istanza , dovremmo pagarlo. Ora proviamo a creare un'istanza con questo modello di istanza specifico. Tutto quello che devo fare è fare clic su questo e dire Crea VM. E nella pagina successiva, vedresti che tutti i dettagli che abbiamo configurato in precedenza sono precompilati. Puoi utilizzare la regione e la zona. Tuttavia, vedresti che tutte le cose come la configurazione della macchina, il disco di avvio sono precompilate con i valori che abbiamo scelto durante la creazione del modello. Puoi vedere che il firewall è configurato automaticamente per consentire il traffico HTTP, e se fai clic su gestione, vedresti che lo script di avvio è già popolato qui. Puoi personalizzarlo ulteriormente se lo desideri, ma idealmente ciò che devi fare è utilizzare un modello di istanza e quindi procedere direttamente e creare un'istanza. Puoi vederlo ora. È in corso la creazione di una nuova istanza utilizzando il modello di istanza. Ora puoi utilizzare questo modello di istanza per creare tutte le istanze che desideri. Torniamo alle istanze VM. Non ho più bisogno dello script di avvio della mia VM, quindi andiamo avanti ed eliminiamolo. Quindi l'ho selezionato e andiamo avanti ed eliminiamolo. Quindi questo eliminerebbe lo script di avvio della mia VM. L'unica VM che sarebbe in esecuzione in questo momento è quella che è stata creata utilizzando il mio modello di istanza con lo script di avvio. Puoi vedere che il nome di quella specifica istanza è il nome del modello aggiunto con un numero, quindi qui c'è il trattino uno e puoi vedere che a questo è assegnato un IP esterno. Facciamo clic su questo e puoi vedere che "Hello World" proviene dal mio modello di istanza con lo script di avvio uno e l'IP interno, 10-1-28-0-4. Assicuriamoci che questa sia l'istanza giusta. Sì.

Negli ultimi passaggi abbiamo visto come possiamo semplificare la creazione di istanze VM e anche impostare alcuni software su di essa. Abbiamo iniziato con l'approccio manuale in cui abbiamo creato un'istanza VM. Ci siamo (indistinti) dentro e poi abbiamo impostato il software usando alcuni comandi. Successivamente, abbiamo utilizzato uno script di avvio e ora abbiamo un modello e ora con il modello è molto semplice. Puoi utilizzare il modello e avviare nuove istanze a tuo piacimento. Sono sicuro che ti stai divertendo e ci vediamo al passaggio successivo.
-: Bentornato. Nei passaggi precedenti, abbiamo visto come semplificare la creazione di una macchina virtuale e l'impostazione del software su di essa utilizzando un modello di istanza. Tuttavia, c'è un altro problema con l'approccio che abbiamo adottato. L'installazione di patch e software del sistema operativo all'avvio delle istanze VM aumenta il tempo di avvio. Quindi, ogni volta che creiamo un'istanza, se stai installando molto software o molte patch del sistema operativo, ci vuole molto tempo per avviare quell'istanza VM, come possiamo evitarlo? Uno degli approcci che possiamo adottare è creare un'immagine personalizzata con le patch del sistema operativo e il software preinstallato. Così possiamo creare un istante, possiamo creare un'immagine personalizzata da essa e possiamo creare ulteriori istanze utilizzando quell'immagine personalizzata. In Google Cloud, puoi effettivamente creare un'immagine personalizzata in diversi modi. Un po' più avanti nel corso, esamineremo cos'è un disco persistente e cos'è un'istantanea. Ogni volta che colleghiamo un disco rigido con l'istanza, ciò che stiamo collegando è un disco persistente e uno snapshot non è altro che una copia di un disco persistente. Puoi creare un'immagine personalizzata da un'istanza VM, un disco persistente, un'istantanea, un'altra immagine o poco dopo parleremo anche dell'archiviazione cloud. Il cloud storage è un archivio di oggetti fornito da Google Cloud. Puoi anche archiviare un file nel cloud storage e creare un'immagine da esso. Ne parleremo un po' di più quando parleremo di disco persistente e snapshot. Per ora, ciò che faremmo sarebbe creare effettivamente un'immagine da un'istanza di Compute Engine esistente. Dopo aver creato un'immagine personalizzata, puoi effettivamente condividerla tra i progetti. Google Cloud fornisce anche la funzione per ritirare le vecchie immagini. Quindi, ogni volta che crei un'immagine, e diciamo che per un periodo di tempo è diventata una vecchia immagine e non vuoi che nessuno la usi, allora puoi contrassegnarla come deprecata. Oltre a contrassegnarlo come deprecato, puoi anche dire che questa è la nuova immagine consigliata. Quindi la raccomandazione è ogni volta che crei una nuova immagine personalizzata e quella vecchia è obsoleta, vai avanti e contrassegna quella vecchia come deprecata. Uno dei vantaggi della creazione effettiva di un'immagine personalizzata è garantire che tutti gli standard di sicurezza aziendali siano incorporati nell'immagine. Quello che puoi fare è creare effettivamente un'istanza VM che aderisce a tutti i tuoi standard di sicurezza aziendale e da ciò puoi effettivamente creare un'immagine. Questo processo è chiamato rafforzamento di un'immagine, creando un'immagine che aderisce a tutti i tuoi standard di sicurezza. Una volta che hai un'immagine indurita, puoi usarlo per creare istanze VM sicure. La prossima raccomandazione è di preferire l'utilizzo di un'immagine personalizzata agli script di avvio. Gli script di avvio includono il tempo di avvio e pertanto si consiglia di creare effettivamente un'immagine personalizzata. Vediamo ora come creare un'immagine personalizzata e come utilizzarla in un modello di istanza. Ho un'istanza VM, che è in esecuzione in questo momento, questa è VM, il mio modello di istanza con script di avvio uno. Se lo riduco al minimo ed espando effettivamente lo spazio di archiviazione, è qui che puoi vedere i dischi. Quindi questi non sono altro che il disco rigido associato alla tua macchina virtuale. Ogni volta che crei una macchina virtuale, il sistema operativo della macchina virtuale viene eseguito su un disco rigido collegato a quella specifica macchina virtuale. Ed è quello che vedi qui, puoi vedere questo disco specifico che è collegato alla nostra macchina virtuale. Puoi vedere che ha anche lo stesso nome: my instance template with startup script one. E puoi vedere che per impostazione predefinita ha una dimensione di circa 10 GB. Ora vogliamo creare un'immagine, vogliamo creare una copia di questo disco specifico. Come possiamo creare un'immagine dal disco specifico? Il modo in cui puoi farlo è andando su azioni e facendo clic su Crea immagine. Quindi andiamo avanti e diciamo crea immagine, Puoi vedere che l'origine di questa immagine specifica è un disco, il nome del disco è presente qui. E se scorri verso il basso, puoi anche scegliere la posizione. Vuoi che questa immagine venga archiviata a livello multiregionale o regionale? Se prevedi che questa immagine venga utilizzata in altre regioni, puoi scegliere multiregionale. Se ti aspetti che questa immagine venga utilizzata solo in una regione specifica, puoi scegliere regionale. Quello che farei è lasciarlo al multiregionale predefinito. Prenderò l'impostazione predefinita per il resto delle cose, inclusa la crittografia, e andrei avanti e direi create. Quando vado avanti e dico crea, c'è un errore che si apre. Il motivo dell'errore è dovuto al fatto che il disco è ora collegato a un'istanza in esecuzione. La raccomandazione è di non creare mai un'immagine da un disco connesso a un'istanza in esecuzione. Dovresti sempre arrestare l'istanza e quindi creare un'immagine da quel disco specifico. Quindi quello che facciamo è effettivamente entrare e andare alle istanze della macchina virtuale, aprirlo in una nuova scheda. E puoi entrare qui e selezionare questo, e vorrei davvero fermarlo. Quindi stiamo arrestando l'istanza della macchina virtuale, aspettiamo che venga arrestata. È possibile selezionare questa casella di controllo e acquisire un'immagine senza arrestare l'istanza. Tuttavia, questo non è davvero raccomandato

e questo è il motivo per cui abbiamo iniziato a fermare l'istanza. E puoi vedere che l'istanza è ora interrotta, e io andrei avanti e ora direi di prenderne una copia. Diciamo creare. Spiacenti, non sta scoprendo il fatto che è stato interrotto, quindi eseguiamo un annullamento. Torniamo ai dischetti e proviamo ora a creare nuovamente l'immagine. Quindi diciamo crea immagine, la chiamerò la mia immagine personalizzata, sceglierò le stesse opzioni che abbiamo scelto in precedenza e dirò crea. Ora, poiché le istanze si interrompevano, usciva e creava quell'immagine specifica. La creazione dell'immagine richiederebbe un po' di tempo, quindi va all'elenco delle immagini, puoi vedere tutte le immagini pubbliche che sono presenti qui, quella che abbiamo appena creato è un'immagine privata. Se aspetti qualche minuto e fai un aggiornamento, vedresti la nostra immagine specifica apparire qui. Puoi vedere che la mia immagine personalizzata è presente qui e in un paio di minuti vedrai che lo stato è pronto per l'uso. Ora posso andare avanti e usarlo per creare un'istanza VM. Quindi quello che dovrei fare ora è andare avanti e dire creare un'istanza, oppure l'altra opzione è in realtà che posso creare anche un modello istantaneo basato su questa immagine specifica. Quindi posso entrare nelle macchine virtuali e andiamo al modello istantaneo e cosa vorremmo fare è utilizzare effettivamente questo modello istantaneo e crearne una copia. Quindi prenderò la copia del modello di istanza esistente. Quindi dirò copia, quindi copierebbe tutto dal modello di istanza precedente. Ora l'ho rinominato nel mio modello di istanza con immagine o direi con immagine personalizzata. E vorrei entrare qui, prenderei la stessa famiglia di macchine di prima. Il disco di avvio, vorrei cambiarlo, quindi invece di Debian, vorrei usare la mia immagine personalizzata. Quindi vorrei entrare qui e fare clic su immagini personalizzate e selezionare l'immagine. Quindi, se aspetti un po', la mia immagine personalizzata dovrebbe apparire qui e puoi dire seleziona. E una volta selezionato, puoi entrare nella gestione della rete di sicurezza del disco e cose del genere. E qui in questo script di avvio, non hai davvero bisogno di fare l'installazione. Tutto ciò che devi fare è creare il file HTML perché vorresti inserire informazioni specifiche sull'host . E una volta creato il file HTML, vorresti anche avviare il server, quindi vorresti avviare Apache. Il modo in cui possiamo farlo è semplicemente dicendo service Apache to start. Quindi stiamo popolando il file HTML e quindi stiamo avviando il server web. Non abbiamo più bisogno di installare Apache perché fa già parte dell'immagine. Quindi questo creerebbe un modello istantaneo, questo sta usando l'immagine personalizzata che abbiamo appena creato. E utilizzando questo modello istantaneo, puoi effettivamente avviare tutte le istanze che desideri e queste istanze verrebbero create con l'immagine personalizzata che abbiamo creato. Posso andare avanti e creare un'istanza utilizzando questa particolare macchina virtuale. Quindi creerò anche una macchina virtuale, prenderò tutti i valori predefiniti suggeriti e dirò create. E mentre la nuova istanza viene creata, ciò che farei è eliminare la vecchia istanza che è stata creata con questo script di avvio. Quindi andrò a cancellarlo, non ne ho più bisogno. Il mio modello di istanza con l'immagine personalizzata uno è ora pronto, dagli un paio di minuti e poi vai avanti e fai clic su IP esterno. E poi dovresti vedere il mio modello istantaneo con l'immagine del cliente uno 10.12805, e puoi vedere che questo è l'IP interno di questo specifico. Quindi puoi vedere che in tutti gli ultimi passaggi abbiamo capito molto sulle macchine virtuali. Stavamo giocando con il ciclo di vita delle macchine virtuali che è fermarle, avviarle, cancellarle. Abbiamo anche giocato con modelli istantanei e abbiamo anche giocato con le immagini.

-: Bentornato. In questo passaggio, proviamo a capire come eseguire il debug dei problemi in caso di problemi con l'installazione del server Web Apache o con l'esecuzione di questo URL specifico. Ora, supponiamo che tu abbia eseguito tutti i passaggi ma non sei in grado di vedere l'avvio di questo URL. Hai utilizzato l'IP esterno, che è presente per l'istanza. E diciamo che lo stai eseguendo e hai ancora un problema. Quali sono le cose che vorresti controllare? Numero uno, assicurati di utilizzare l'IP esterno dell'istanza in esecuzione. Se aggiorno la pagina VM, è qui che posso vedere l'IP esterno. Assicurati di fare clic su di esso e di aprirlo in una nuova finestra. Quindi questo è il numero uno. Quindi assicurati di utilizzare l'URL corretto. Se sei sicuro di utilizzare l'URL corretto, allora quello che puoi fare è andare e SSH in esso. Assicurati di essere in grado di accedervi tramite SSH. Se sei in grado di accedervi tramite SSH, assicurati che il tuo server Web Apache sia installato correttamente. Posso fare un LS e guardare /ware/www/html. Quindi ls/ware/www punto /html. E cosa vedo? Vedo un file index.html, che è presente qui. Ora, vorrei vedere qual è il contenuto presente in questo file specifico. Quindi dirò cat/ware/www /html/index.html. Cosa mostrerebbe? Mostra il contenuto di questo file specifico. Puoi vedere che il contenuto di questo file specifico è quello che vedo sul browser web. Quindi assicurati che questo file sia presente e abbia il contenuto giusto che vorresti vedere. Dopo esserti assicurato che il file sia presente e che anche il contenuto sia a posto, quello che puoi fare è anche assicurarti che il servizio Apache sia avviato correttamente. Quindi puoi digitare Service Apache 2 space start. Prima di poter avviare un servizio Apache 2, avrei effettivamente bisogno di creare uno pseudo alimentatore. Quindi farei uno pseudo psu e poi farei un servizio, Apache 2 start. E questo garantirebbe che anche il server Apache sia avviato. E dopo questo puoi effettivamente andare a controllare se il tuo URL funziona. Se il tuo URL continua a non funzionare, c'è un'altra cosa che puoi controllare per fare clic su questa istanza VM. Quindi, quando l'istanza VM è attiva e se scorri più in basso, dovresti vedere i firewall. E nei firewall

dovresti vedere un traffico HTTP (indistinto) controllato. Se non lo vedi selezionato, puoi effettivamente modificarlo e controllarlo. Quindi, se dici modifica, puoi entrare qui, selezionare questa casella di controllo e assicurarti di salvarla. Una volta salvata l'istanza VM, puoi fare clic sull'IP esterno di essa e dovresti vedere la pagina apparire. In questo video abbiamo visto come eseguire il debug dei problemi con l'installazione del server Web Apache su un'istanza VM. Sono sicuro che ti stai divertendo e ci vediamo nel passaggio successivo.
-: Bentornato. Facciamo un passo indietro e osserviamo l'intera interfaccia offerta dalla console web di Google Cloud Platform. Questo è anche chiamato solo la console. Quando tutte le persone dicono console, allora si riferiscono a questa cosa specifica. E qui, la cosa più importante è la matita che è presente qui. Puoi entrare qui e accedere ai diversi servizi che sono presenti qui. L'altro modo in cui puoi accedere ai servizi è il modo in cui abbiamo fatto fino ad ora. Quindi diciamo che vorrei accedere a Kubernetes Engine. Posso digitare Kubernetes qui. Oppure posso andare alla matita e posso navigare verso Kubernetes. Sarebbe sotto "Compute", e poi posso entrare e accedere a quel servizio specifico. La cosa interessante dell'uso della matita è che ogni volta che vuoi effettivamente contrassegnare qualcosa come preferito, puoi effettivamente appuntarlo. Forse giocherò molto con Compute Engine, quindi posso andare avanti e fissare questo. Quindi quello che accadrebbe è che "Compute Engine" verrebbe fuori qui. Quindi, quando faccio clic sul mio menu di navigazione, vedo "Compute Engine" proprio in alto. Quindi, se stai giocando molto con qualsiasi servizio, puoi utilizzare questa funzione e accedere direttamente alle tue istanze VM. Ora, vediamo cosa c'è sotto "Casa". Sotto "Home" puoi vedere "Dashboard". che abbiamo visto quando siamo entrati per la prima volta in Google Cloud. Contiene le informazioni sul progetto. Contiene diverse API che stiamo utilizzando. Puoi vedere che stiamo utilizzando molte chiamate API. Ogni volta che creiamo risorse, vengono chiamate molte API e queste sono le risorse che stiamo vedendo qui. Qui puoi anche vedere lo stato attuale della piattaforma Google Cloud. Puoi vedere che tutti i servizi sono normali in questo momento. Puoi anche vedere le informazioni di fatturazione e, se scorri verso il basso, vedrai anche molte informazioni sul monitoraggio e la segnalazione degli errori. Un'altra cosa che vedresti qui è l'attività. Se vai su "Attività", vedresti molte informazioni su ciò che hai fatto fino ad ora. Quindi qui, puoi vedere che ho svolto molte attività creando ed eliminando VM e giocando con loro. Quindi puoi vederlo. E se vuoi, puoi andare avanti e filtrare tra le attività. Quindi puoi andare al tipo di attività e selezionare il tipo di attività a cui sei interessato. Configurazione, accesso ai dati, sviluppo o monitoraggio. Puoi anche scegliere su quale tipo di risorse vuoi concentrarti. Puoi vedere che quasi 75 diversi tipi di risorse sono selezionati qui per impostazione predefinita. Puoi scegliere quello che vorresti guardare. L'ultima scheda, presente qui, è "Consigli". Google Cloud analizzerebbe automaticamente il tuo utilizzo del cloud e, se hanno consigli, li mostrerebbero qui. Abbastanza strano, come puoi vedere, non ci sono raccomandazioni per me. A parte questo, se fai clic sulla matita e vai sul lato sinistro, puoi vedere tutti i servizi che vengono presentati qui, giusto? Quindi puoi vedere i servizi di calcolo, puoi vedere i servizi di calcolo sul lato sinistro. Molti di loro qui. Puoi anche vedere diversi, puoi vedere servizi di archiviazione, servizi di database. Puoi vedere i servizi di rete che sono presenti qui, e ci sono anche altri servizi per supportare le tue operazioni e ci sono alcuni strumenti e vedrai anche servizi per aiutarti con i big data e l'intelligenza artificiale. Come puoi vedere qui, questo è il progetto su cui stiamo lavorando. Il mio primo progetto. Se lo desideri, puoi effettivamente creare un nuovo progetto e creare risorse anche nell'ambito di quel progetto specifico. Per ora, continuiamo a utilizzare il mio primo progetto che è il progetto predefinito che è stato creato per noi. Oltre a questo, nel menu in alto, puoi anche vedere lo stato della tua prova gratuita. Se fai clic su questo pulsante, puoi vedere la quantità di credito di prova gratuita rimanente e il numero di giorni rimanenti per la prova gratuita. Puoi ottenere assistenza facendo clic su questo punto interrogativo qui e, se ci sono notifiche, puoi vederle qui. Vengono create notifiche ogni volta che crei , elimini o aggiorni un'istanza. Ci saranno notifiche create per questo. E puoi fare clic su questi tre punti qui per vedere alcune altre impostazioni. Se entri qui e vai alle impostazioni del progetto, sarai in grado di vedere le impostazioni per il nostro progetto attuale. L'idea alla base di questo passaggio era quella di navigare rapidamente e giocare con la console cloud in modo da acquisire familiarità con essa quando si continua a utilizzarla durante questo corso specifico. Quello che farò ora è andare avanti e fai clic su chiudi qui e ci vediamo nel passaggio successivo.
Istruttore: Bentornato. In questo passaggio, esaminiamo i pochi scenari relativi a Compute Engine. Iniziamo con il primo. Quali sono i prerequisiti per poter creare un'istanza VM? Fino all'amore in questo corso, stiamo utilizzando il progetto predefinito che è presente qui. Abbiamo utilizzato il mio primo progetto. Questo viene creato per impostazione predefinita per primo. Poco dopo vedremo anche come si possono effettivamente creare nuovi progetti. La seconda cosa importante è un account di fatturazione. Ogni volta che esegui il provisioning delle risorse nel cloud, devi pagarle e in GCP paghi per le tue risorse utilizzando un account di fatturazione. Quindi hai un progetto e il progetto dovrebbe essere associato a un account di fatturazione. In precedenza, quando stavamo creando un account Google Cloud Platform, abbiamo fornito i dettagli della nostra carta di credito e tali dettagli vengono utilizzati per creare un account di fatturazione per noi. Se digiti l'account di fatturazione qui, puoi passare all'account di

fatturazione, che è già presente. Quindi vedresti che esiste un account di fatturazione predefinito che è già stato creato per noi. Si chiama account My Billing. E puoi vedere la panoramica dell'account di fatturazione e anche la panoramica dei pagamenti qui. Se sei in una prova gratuita, molto probabilmente il tuo saldo sarebbe pari a zero, e puoi anche vedere la carta che hai configurato qui. La terza cosa importante è che le API di Compute Engine dovrebbero essere abilitate. Ogni volta che utilizzi un servizio Google Cloud, le relative API devono essere abilitate prima di creare risorse utilizzando quel servizio specifico. Se creo una macchina virtuale di Compute Engine, le API di Compute Engine devono essere abilitate. Se digito le API di Compute Engine, puoi dire che questa è l'API di Compute Engine che compare. Se vado lì, l'abbiamo già abilitato quando stavamo creando la prima macchina virtuale. Ora, se vai a un nuovo progetto, se crei un nuovo progetto e ci vai, vedresti che l'API non sarebbe abilitata per impostazione predefinita. Prima di creare la macchina virtuale, dovresti abilitarla. Per questo progetto specifico l'abbiamo già abilitato, quindi puoi entrare e dire gestisci. E qui puoi vedere i dettagli su questa specifica API. Puoi vedere che questa è l'API di Compute Engine e c'è un nome di servizio specifico che è presente per questo. Questo è compute.googleapis.com e viene utilizzato per creare ed eseguire macchine virtuali sulla piattaforma Google Cloud. Questo è ciò che abbiamo fatto e puoi vedere l'utilizzo di quella specifica API qui. Puoi vedere che ho iniziato a registrare questo corso una settimana fa, e puoi vedere tutte le informazioni al riguardo qui. I tre prerequisiti per poter creare qualsiasi risorsa, il numero uno è che hai bisogno di un progetto, hai bisogno di un account di fatturazione in modo da poter fatturare le tue risorse. Il numero tre sono le API per quella specifica risorsa. Creerò un'istanza VM qui. Quindi l'API Compute Engine dovrebbe essere abilitata. Successivamente, ti serve un hardware dedicato per le tue esigenze di conformità, licenza e gestione. In genere ogni volta che creiamo una macchina virtuale, viene creata su un hardware condiviso. Quindi l'host su cui viene creata la tua macchina virtuale potrebbe essere condiviso da più clienti. In alcuni scenari, potresti volere hardware dedicato. Su quell'hardware specifico, vuoi che siano presenti solo macchine virtuali della tua azienda. Come lo chiedi? Il modo in cui puoi richiederlo è entrare in un unico inquilino. Quindi, se cerchi solo un unico inquilino. Oops, l'unico inquilino non parla di niente. Passiamo a Compute Engine. Quindi torniamo a Compute Engine. E qui puoi vedere, nodi unico inquilino. Nodi single-tenant. È qui che puoi entrare e puoi effettivamente creare un unico nodo tenant. La prima cosa che devi fare è creare un gruppo di nodi. Quindi puoi creare un unico gruppo di nodi tenant e qui puoi dare un nome a questo gruppo di nodi. Sceglierò una delle regioni che è presente qui. Non creeremo comunque questo gruppo di nodi, quindi non ha molta importanza. Puoi scegliere una qualsiasi di queste regioni che sono presenti lì. La prossima cosa che devi configurare è il modello di nodo. Non abbiamo ancora un modello di nodo, quindi puoi entrare e dire crea un modello di nodo e crea un modello di nodo qui. Nel modello di nodo è dove specifichi il tipo di nodo che desideri. Quindi posso scegliere N1 o N2, per esempio. Posso scegliere se desidero un SSD locale. Posso scegliere se vorrei un acceleratore GPU. La parte importante qui sono le etichette di affinità. Ogni volta che crei una nuova istanza di macchina virtuale e desideri che venga distribuita a questo gruppo di nodi specifico, dovrai assegnarle un'etichetta di affinità corretta. Quindi puoi dire aggiungi etichetta di affinità e dire che questo è il gruppo A e puoi dire crea. Quando provo a creare un modello di nodo, ottengo un'altra cosa. Il gruppo di etichette di affinità nodo viola i vincoli di formato. Dovrebbe contenere solo caratteri minuscoli. Quindi andiamo a creare questo gruppo. Lo stesso è il caso dei valori. Quindi andrò a cambiarlo anche in A. Quindi raggruppa A e dì crea. Ora questo creerebbe il nostro modello di nodo e possiamo usare quel modello di nodo nel nostro gruppo di nodi per creare un gruppo di nodi. Dopo aver creato un modello di nodo, puoi dire continua. E se lo desideri, puoi aggiungere il ridimensionamento automatico e specificare il numero di nodi che desideri. Per ora, diciamo off, e diciamo solo un nodo. Non lo creeremo perché sono molto costosi. Dopo aver configurato il ridimensionamento automatico, puoi configurare tutte le impostazioni di manutenzione. Ora non creerò questo nodo tenant unico. Quello che farei è annullare. Ora come puoi creare macchine virtuali per quel gruppo di nodi specifico? Il modo in cui puoi farlo è andare alle istanze VM, creare. Questo è esattamente ciò che facciamo di solito per creare istanze di macchine virtuali. L'unica differenza qui sarebbe andare alla sicurezza della gestione, questa rete e l'affitto esclusivo. E qui puoi specificare l'etichetta di affinità. E qui nell'etichetta di affinità nota c'è il punto in cui possiamo soddisfare il gruppo A. Quindi qui puoi vedere il valore dell'operatore chiave. Quindi possiamo dire gruppo in A, simile a questo. Quindi qui dice il carico di lavoro nel front-end. Quindi carico di lavoro nel front-end, oppure puoi specificare il gruppo e dire il valore del gruppo, che è A. Ecco come puoi specificare l'etichetta di affinità del nodo. Quello che accadrebbe ora è se hai il nodo con questa specifica etichetta di affinità già creata, l'istante VM verrà creato su quel nodo specifico. Quindi, se desideri un hardware dedicato per i tuoi reclami, le tue esigenze di licenza e gestione, puoi scegliere un nodo single tenancy. Prossimo scenario, ho migliaia di macchine virtuali. Voglio automatizzare la gestione delle patch del sistema operativo, la gestione dell'inventario del sistema operativo e la gestione della configurazione del sistema operativo. Ad esempio, vorrei gestire questo software installato su migliaia di macchine virtuali. Come lo posso fare? Se torno a Compute Engine e riduco a icona i gruppi di istanze di archiviazione delle macchine virtuali? Qui è dove puoi vedere

il gestore VM. Qui è dove puoi effettivamente eseguire la gestione delle patch. Quindi, come puoi vedere qui, benvenuto in VM manager. VM Manager è una serie di strumenti che possono essere utilizzati per gestire i sistemi operativi per parchi macchine virtuali di grandi dimensioni. Quindi, se vuoi gestire migliaia di VM e automatizzare la gestione delle patch, l'opzione da utilizzare è VM manager. Ora supponiamo che tu voglia accedere alla tua specifica istanza VM e che tu voglia installare del software. Ad esempio, server Web Python o Tom Cat o server Web Apache. Come si fa a farlo? Il modo in cui puoi farlo è SSH in esso. Lo abbiamo fatto all'inizio del corso. Non vuoi esporre una macchina virtuale a Internet. Come possiamo farlo? Poco dopo, quando parleremo di VPC, parleremo delle regole del firewall e diversi tipi di cose. Ma per ora, l'opzione più semplice è non assegnare un indirizzo IP esterno. Una volta che un'istanza VM non ha un indirizzo IP esterno, non sarà accessibile da Internet. Vuoi il traffico HTTP verso la tua VM. Come puoi farlo? L'abbiamo già guardato. Puoi configurare le regole del firewall. Dove possiamo configurare le regole del firewall? Puoi andare alle istanze VM della macchina virtuale e qui quando stiamo creando la macchina virtuale è quando possiamo effettivamente dire consentire il traffico HTTP o consentire il traffico HTTPS. Quindi, se la tua istanza VM è un server Web e desideri consentire il traffico HTTP o il traffico HTTPS, qui è dove puoi effettivamente configurare le regole del firewall. Nel passaggio, abbiamo esaminato alcuni degli scenari importanti relativi a Compute Engine. Sono sicuro che ti stai divertendo e ci vediamo nel passaggio successivo.

04 - Instance Groups and Load Balancing for Professional Cloud Architect

Welcome back. Having played with virtual machines, it's time to play with a number of them. How do you create a group of virtual machines? That's where we would be talking about instance groups. And how do you load balance between multiple virtual machines? That's where load balancing comes into picture. In this step, we'll be looking at instance groups. We'll be looking at two types of instance groups, managed instance groups and unmanaged instance groups. And we'll also look at Cloud load balancing. There are a number of load balancing options that are provided by Google Cloud. We'll explore the different features of load balances. We'll understand how to choose a load balancer in GCP and we'll also see how to set up a microservices architecture. You have multiple microservices. There might be multiple versions of each microservice and you'd want to deploy these microservices in multiple regions. We'll see how you can implement that using load balancing and multiple managed instance groups. I'm really excited to get you started with instance groups and load balancing. And I'll see you in there.

Instructor: Welcome back. Welcome to this new section on Instance Groups. In this section let's see how you can use Instance Groups to manage a group of virtual machine instances. What are instance groups? How do you create a group of virtual machine instances? That's where we would go for an instance group insurance group is nothing but a group of virtual machine instances managed as a single entity. You can manage group of similar VMs having similar life cycle as one unit. There're two types of instance groups that you can create. The first one is Managed Instance Group. You'll also see it referred as MIG Managed Instance Group or MIG. These are identical VMs created using a template so these VMs, which are part of a Managed Instance Group have the same machine type, same image and the same configuration. So they're created using a instant template and some of the features that are provided for the Managed Instance Groups include auto-scaling based on the number of users using the instance group you can scale the number of compute instances up and down. Managed instance groups also provide auto healing. You can configure a health check and if the health check fails that specific instance would be automatically replaced with a new instance. Another important feature is managed releases. You can go from one version to another without any downtime. The other type of instance groups are Unmanaged Instance Groups. In unManaged Instance Groups you can have VMs with different configurations. You might have VMs with different images which were used to create them or you might have VMs with different hardware so you don't want to group VMs with different kinds of configurations. In those kind of situations you create an unManaged Instance Group. The important thing to remember is that with unManaged Instance Groups, you do not get any of the features like auto scaling or auto healing. Only the Managed Instance Groups. That's basically identical VMs which are created using an instance template only for them, you have features like auto scaling and auto healing for unmanaged instances. You do not have the features of autoscaling and auto healing and important thing to remember is that unmanaged instances are not recommended unless you need different kinds of VMs. If you want to create multiple instances of the same kinds of VM in those kind of situations always go for a Managed Instance Group. Go for unmanaged only when you need different kinds of VMs. The location of an instance group can be either zonal or regional. You can create it in a specific zone or you might want the instance group to distribute the instances across different zones in a specific region. Obviously regional is recommended because it gives you higher availability. Let's now talk about the most important type of instance group, which is Managed Instance Group. As we discussed earlier, it's also referred to as MIG, Managed Instance Group. Manage Instance Group is an identical set of VMs created using an instance template. So you have an instance template which has the configuration and you can use that to create a Managed Instance Group. And the important features we have already referred to some of them. The important features of a Managed Instance Group are, it can maintain a certain number of instances. You can say I would want two virtual machine instances running all the time. In that kind of situation even if one of the instances fails the Managed Instance Group will replace that with a new instance, so it can maintain a certain number of instances if an instance crashes, MIG launches another instance. It can detect application failures using health checks so it's self healing. So basically you can configure a health check on the application which is deployed on your instance and if the health check fails that instance would be replaced by a new instance. You can also increase and decrease the number of instances that are part of your Managed Instance Group based on the load, based on the number of users using the application. So it provides auto scaling. You can also add a load balancer attached to your Managed Instance Group to distribute load among the different virtual machines that are part of your Managed Instance Group. And if you're creating regional Managed Instance Groups then you can distribute the instances across multiple zones. This gives you higher availability so regional Managed Instance Groups provide you with higher availability compared to zonal Managed Instance Groups. The other important feature of a Managed Instance Group is that you can release new application versions without downtime. Whenever I have a new application version I would create a new instance of the instance template and I can upgrade from one instance of instance template to another instance of instance template without any downtime. Among the features which are provided are rolling updates. Basically you can release new version step by step. You can update a percentage of instance to the new version at a time. If you have 10 instances, you can update them two at a time for example. The other option you also have is canary

redeployment. Basically, you can test the new version of the instance template with a group of instances. So out of the 10 instances which are live right now you can choose two instances. I can test it with those two instances and then I can roll it out to all the other instances at one time. This is called canary redeployment, so with Managed Instance Groups, you can do rolling updates and canary redeployment as well. You can update a group of instances at a time or you can test with a group of instances and then roll it out at one time to all the remaining instances. In this step we got introduction to the Instance Groups. If you want to manage a group of virtual machines together you would create an instance group. There are two types which are managed and unmanaged in a Managed Instance Group. It's all identical VMs and a Managed Instance Group provides you a number of features, auto scaling, self healing, and managed releases. When you want different kinds of VMs in a group then you can go for Unmanaged Instance Group. An Unmanaged Instance Group on the other hand does not provide you with self healing or auto scaling. I'm sure you're having a wonderful time and I'll see you on the next step.
Instructor: Welcome back. In the last step we got an introduction to a managed instance group. In this step, let's look at what is involved in creating a managed instance group. The first thing that you need is an instance template. Instance template is mandatory, and once you configure the instance template you need to configure autoscaling to automatically adjust the number of instances based on the load. So you can configure minimum number of instances, maximum number of instances. You can also configure autoscaling metrics. How do you want to do autoscaling? You want to do it based on CPU utilization or do you want to do it based on load balancer utilization? Or there is also a monitoring tool which is provided by Google Cloud. It is called Stackdriver. So do you want to use any metrics from Stackdriver to do the autoscaling. So you can configure all that when you're configuring autoscaling. In addition to that you can also configure a cool down period. If I'm executing an autoscaling action right now, how much time should I wait for before looking at the autoscaling metrics again? That is what is configured by the cool down period. This would ensure that scaling up and scaling down does not happen very, very frequently. So you perform a scaling action, so you increase the number of instances by three and then wait for some time before you'd look at the metrics again. That is what is configured by the cool down period. You can also configure a few scale-in controls. You don't want a sudden drop in the number of VM instances. In those kinda situations, you can configure scale-in controls. You can say, I don't want to scale-in by more than 10% or three instances in five minutes. So in that interval the maximum number of instances that should be scaled in or the maximum number of instances that should be reduced should be 10% or you can also configure a specific number like three. I only want to reduce three instances in five minutes. As part of your autoscaling configuration you can also configure the settings for auto healing. Auto healing is just the process where the managed instance group can identify unhealthy instances and replace them with healthy instances. How do you do that? The way you can do that is by configuring a health check. Along with the health check you can also configure an initial delay. Whenever you're starting up a new virtual machine you need to give it some time before it's ready to accept traffic. You can configure an initial delay saying how long should you wait for the app to initialize before running a health check? Now we have a lot of theory that we talked about until now, let's now get started with a demo on the managed instance group. I have one VM instance, which was manually created earlier so I actually used the instance template that we had earlier. So we have this instance template my-instance-template with custom image, which we created earlier in the last section. I used it and said, Create VM and I created the VM with the default settings. Now, what do we wanna do is to create instance groups. Where do we create instance groups? I can minimize virtual machines and go to instance groups in here. And you can see that as part of instance groups, there are instance groups and health checks. I'll go to instance groups first. This is where you can create your instance groups. Instance groups lets you organize your VM instances or use them in load balancing backend service. You can group existing instances or create a group based on instance template. If you're grouping existing instances, then most probably you're creating an unmanaged instance group. If you're creating a group based on an instance template you are creating a managed instance group. Let's go ahead and create an instance group. Once the page loads up, you'd be able to see that on the left hand side you would see the different types of instance groups that you can create. As we discussed earlier, there are two types of instance groups that you can create, managed and unmanaged. In managed instance groups, there are two types, stateless and stateful. As we discuss earlier, when we are creating managed instance groups we are creating identical VMs which are created using a template. However, you could go for unmanaged instance groups when you'd want different configuration for different VMs in the same group. So if you go to unmanaged instance group and choose a location and choose a network as well. Let's choose the default network. You'd see that you'd be able to select the different VM instances that you'd want to be part of this specific unmanaged instance group. If I have 10 different VMs which are previously created, I can choose them to be part of this unmanaged instance group even if they have different configuration. So in an unmanaged instance group, you can have different VMs with different kinds of configuration. However, this is very, very infrequently used. The most frequently used kind of instance groups are managed instance groups. You can see that in the case of managed instance groups there are two varieties in here, stateless and stateful. If you're creating something like a database then that is a stateful workload. You'd want to

ensure that all the data is persisted. You don't want any of the instances to be removed without the data being persisted. And when you are running stateful workload like databases, you would go for a managed instance group stateful. Typically, we would run web applications or REST API which do not have any state, and that's where we would go for stateless managed instance groups. Over here I go ahead and give the name my managed instance group. The first thing that you need to select is a instance template. What instance template should I use to create the instances in the managed instance group? I would choose my instance template with custom image. This is the most recent one that we created in one of the previous steps. After that, you can choose the location. You can choose either single zone or multiple zones. Choosing multiple zones will provide you with higher availability. I'll choose the default region and I'll choose the default zones, which are suggested. If you go further down, you can configure autoscaling. There are multiple autoscaling modes, ON, SCALE-OUT, and OFF. OFF is simple, right? You don't want to autoscale at all. You'd want to select autoscaling mode ON when you'd want the managed instance group to automatically add and remove instances from the group. However, if you'd want the managed instance group to only add instances in that kind of situation you can choose SCALE-OUT. Typically, we would go for ON. Over here you can configure the minimum number of instances and the maximum number of instances that you would want. For now, what I would do is I would put minimum number of instances to two and maximum number of instances to two as well. The next thing you can choose is how a managed instance group can decide when to autoscale. That's basically the autoscaling metrics. The default autoscaling metric which is used is CPU utilization. When you click this, you'd see that there are different metrics you can use other than CPU utilization as well. If you attach the managed instance group with a load balancer, you can make use of the load balancer utilization to decide when to autoscale. You can also use metrics from Cloud Monitoring. Cloud Monitoring is a managed service, which is offered by Google Cloud where you get all the metrics from all the other services. You can use one of the Cloud Monitoring metrics to decide when to autoscale as well. I'll choose CPU utilization and I'll leave it at the default one, which is 60% CPU utilization. Predictive scaling is one of the recent features which is added in where you can have Google Cloud look at the historical load. Google Cloud can look at the last week or last few days data and it can use that data to do predictive autoscaling. Let's not really worry about it and I would say done. You also have the feature to autoscale based on a scale. You can say, at this time I would want to increase the number of instances to this specific thing, and at a later point in time I would want to decrease the number of instances. You can also configure a cool down period. Once a autoscaling decision is made, how long do you want to wait before you would look at the metrics again? I'll take the default, which is 60 seconds. You can also configure scale-in controls. Let's say you have a hundred instances and you have a lot of load. Suddenly all the load goes down. You don't want to reduce the number of instances directly to five or ten. That's where you can set in scale-in controls. Over here, if you enable scale-in controls, you can say I don't want to scale-in by more than 10% of VMs in a 10 minute period. So instead of directly going in from hundred instances to 10 instances, you can say I'd want to first go down to 90, then 80, then 70, then 60 and so on and so forth. Let's not really worry about scale-in, so I'll disable that and the next thing you can configure in here is very, very important. That's auto healing. How can a managed insurance group decide if a VM is healthy or not? That's where it makes use of a health check. So you can configure a health check in here. You can say create a health check. I'll call this my VM health check. Remember to change the protocol to HTTP. We are using HTTP URL to access the application which is deployed on our virtual machine. So the protocol should be HTTP, the port we are running the application on is port 80. You can take the defaults for proxy protocol and request paths. These health checks are done on a periodical basis and you can log them as well in Cloud Logging. Cloud logging is a managed logging service in Google Cloud. So all the logs from all the managed services in Google Cloud can be sent to Cloud Logging. I don't really want to worry about logs in here so I leave it at OFF. The last thing you can configure as part of your health check is the health criteria. Let's say I would want to execute my health checks every one minute or every ten seconds. That's what you can configure in the check interval. The default is five seconds. So the health check will be done every five seconds. The next thing you can configure is a timeout. How long to wait before a request is considered a failure. Let's say the managed instance group has sent a request to the VM and it does not get a response for five seconds. In that kind of situation, it would be considered as a failure. So you're configuring the timeout period in here. The next things you can configure are the healthy threshold and the unhealthy threshold. How many consecutive successes must occur to mark a VM instance as healthy? That's the healthy threshold. The unhealthy threshold is the reverse. How many consecutive failures must occur to mark a VM instance as unhealthy? I'll take the defaults for all the health criteria and I'll click save. Once you configure a health check you can also configure a initial delay. Typically, your VM stances need time to boot up. How much time does your VM instance need to get ready? That's what we would configure in initial delay. The default is 300. I think it's too much, so let's go down to 30. So I'll configure a initial delay of 30 seconds. I'll take the defaults for rest of the things and I'll go ahead and say create. This would start the creation of our managed instance group. The creation of the managed instance group would take a little while. What I would recommend you to do is to take a break, grab a coffee, and I'll see you on the other side.

Welcome back. The creation of the Instance group took about 10 minutes and at the end of the 10 minutes I can see the new instance group which is created. You would see a warning in here which says the new number of instances is equal to the maximum number of instances. This means the auto-scaler cannot add or remove instances from the instance group. Make sure that this is a correct setting. That's fine. You don't really need to worry about this warning. The thing which I'm trying to do is to have two instances running all the time with Health Check Enabled. So even if one of the instances is killed you'd see that it would be automatically replaced with a new instance by the managed instance group. In an ideal world probably the maximum number of instances would be higher so probably you would set it to a higher number like 10 or 20 or 30, whatever you expect to be the maximum mode of instances that you might want to use. On the Manage Instance Group page I can see what is the name of it, how many instances what is the template which was used to create it. You can see that this is a managed instance group, when was it created and you can also see the Auto-scaling configuration on the right hand side. Now when I click inside the My Managed Instance group I'll be able to see a few more details. You can see that there are two instances in total and two instances are running. You can see what is the location of it. You can see how many of them are healthy, a hundred percent healthy. And you can see that auto healing is configured with a health check. Where is that Health Check configured? You can actually click it and you'll be taken to the health check and you can see all the details of the health check which are configured. You can see that we are making a HTP request to the root on Port 80. Let's go back to the instance group and over here you can also see the two instances which are part of the instance group. You can see that the name of them is configured by default. So my managed instance group is the name of the instance group hyphen a few random letters and numbers and you can see that these two are created in two different zones, US Central One F and US Central One C. Now I can click in here and this would take me to the configuration of that specific virtual machine instance. Now I'll click on virtual machine instances in here to see all the virtual machine instances which are present in here. This is an instance that I created earlier from Custom Image, so I use the instance template to create this specific image directly, and these two are the ones which we created through the managed instance group. And over here you can see that these two are tagged with in use by a specific managed instance group. And if you click the external IPs of both of these they should be bringing up the HTP response. So hello world from My Managed Group, the name of it, and the IP address, the internal IP and similarly the other one. So this also is working. Now I'll go back to the instance groups and I'll go to the specific instance group. And if you want, you can also edit the group. So you can go here in say edit the group. And over here you can actually configure if you want you can actually change the metrics which are used for auto-scaling. You can change the health check or you can increase or decrease the number of instances. Let's say I would want to actually just have one instance, then I can change this to one as well. Or if I want five instances, I can increase it to five as well. So if I actually, let's say changed minimum to three and maximum to three as well, then and say Save, what would happen? You'd see that this group would be updated. Let's go back and within a little while, if you gave it a few minutes, you would see that the third instance would also be up and running In this step, we got started with playing around with the instance group. There are a lot more things that we'd want to learn about it, and I'll see you in the next step.

-: Now that we understood how to create a managed instance group let's shift our attention to updating a managed instance group. We did one of the basic updates in the last step. We updated the number of instances or we updated this Auto Scaling configuration to Max three and Min three, and we saw that the number of instances would be increased automatically. Other than that, there are a lot of other updates that you can do with a managed instance group. Let's look at them in the specific step. The first type of update that you can do is a rolling update. A rolling update is a gradual update of the instances in an instance group to a new instance template. Let's say I have one version of the instance template and I would want to upgrade to the new version of the instance template. The earlier version of the instance template might be referring to an old version of the application and the new version of the instance template is referring to the new version of the application. In that kind of situation what we are doing in here is switching from an old version of the application to the new version of the application. When we are doing this, the first thing that you need to do is to specify the new template. Optionally, you can also specify a template for canary testing. By default, what would happen is a rolling update. Let's say I have 10 instances I can say I would want to update two instances at a time, so I would want two instances, first, two instances next, two instances after that, so we are updating in a group of two instances each. The other option is to test a set of instances first. So you would configure a new template that is basically called the canary template and you would test two instances. Once the two instances are all tested you would roll out at once to the remaining eight instances. This approach is called a canary testing approach. You're testing a set of instances quickly and then you are rolling it out to all the instances at the same time. You can do both canary testing and a gradual rollout using a update of a managed instance group. The next thing you need to specify is how you'd want the update to be done. When should the update be done? Should the update start immediately? That's basically called proactive, so proactively update all the instances, or you can also be opportunistic when the instance group is resized or when you have the instances are killed then you'd want to replace them with, with an instance created using the new template. Typically, we would go for proactive

but there might be situations where the update is not really important. You can wait for some time. In those kind of situations you can go for opportunistic. The next thing you don't want to configure is how should the update happen? How many instances should be updated at any point in time? That is maximum search. Let's have 10 instances. I can say I would want to add a maximum of two instances at any point in time. So in total there will be 12 instances that would be running at any point in time. You can also configure maximum unavailable. While the update is going on how many instances can be offline? If I have 10 instances and let's say two instances can be offline, then those two instances can be updated to the new version. And the remaining eight instances will be serving traffic. The other type of update that you can do is a rolling restart or replace. This is not really an update, but you'd want to actually let's say restart all VM instances without any downtime or you'd want to replace VM instance with a new version created with the same template. So over here, there is no change in template but you'd want to either restart the existing VM or you'd want to replace the existing VM with a new VM using the same template. Even in this situation, you can configure maximum search, maximum unavailable, and what do you want to do? You can configure whether you want to restart the instances or replace the instances. So where can we configure the rolling update, rolling restart, and replace? Let's go back to our managed instance groups. This is where you can see my manage instance group with three instances. Now go inside. If you want to configure a rolling update you'd want to go for update VMs. So this is where you can go in into update VMs and you can configure a new instance template. So you can say, I would want to change my Instance template from my Instance template with custom image to let's say a new version of it. Let's say if the my Instance template with startup script is the new version of it you can choose the new template. Another option it also provides you with is canary testing option. You can say, "I don't want to immediately shift all instances to the new version. I want to leave the existing version running and I would want to add a new template." I can say I only want to run let's say I would want to choose a new template. Let's say the startup script is a new template. I can go and choose that and I can say I would want to first run one instance with a new template. Once I test it, I can then increase the number of instances for that specific template and decrease the number of instances for the previous version template. So in the case of rolling update we are actually changing the template and we also have the option to do canary testing. We can look at all the other options in the update configuration. You can go to the update configuration and say whether you'd want to do a automatic upgrade. This is called proactive. The other option is opportunistic where you'd want to do the update when VMs are replaced, refreshed, or restarted. If you select automatic you can choose a few other things as well. You can configure what are the actions allowed to update VMs. You want to only replace or only restart or only refresh or use a combination of this. You can also configure if you want to retain the instance names. The other thing in you can configure in here is temporary additional instances. This is also called max search. How many new instances should be first created before the old instances are deleted? This is where you can configure max unavailable. How many available, how many instances can be unavailable during the update? So this is for rolling updates where we are updating a instance template. Let's go back and now we look at rolling restart and replace. So if you click restart or replace VMs this is where you can do rolling restart or replace. In the case of rolling restart or replace we are not changing the instance template. All that we need to do is either restart or replace of existing VM instances using the existing template itself. So you can choose the right operation you want to do. If you want to restart then you can configure only max unavailable. How many instances can be unavailable during the restart? So if there are 10 instances, let's say you'd want to restart two instances at a time, then you can configure that in here. If you want to replace the VMs in that kind of situation you also have the option to configure a max surge. So in addition to maximum unavailable instances you can also configure temporary additional instances, that's basically max surge. So I can say I want to create two new instances before starting the replacement process. Understanding max surge and max unavailable is really, really important. We'll look at a few examples in the next steps as well. In this step we talked about how you can actually update a managed instance group. We talked about rolling updates which are used to gradually update instances in our instance group to a new version of your application or a new version of your instance template. The other option is rolling restart or replace where you want to restart or replace the existing instances of a VM without any change in the instance template. I'm sure you're having a wonderful time and I'll see you on the next step. Welcome back. In the previous section, we created a number of VM instances. Now we would want to load balance between all those instances. How do you do that? That's where we would be going for a cloud load balancing. In this section, let's look at cloud load balancing in depth. A cloud load balancer distributes user traffic across instances of an application in single region or multiple regions. So if you have multiple VM instances in multiple zones and multiple regions, you can distribute traffic between them using a cloud load balancer. This is a fully distributed software-defined manage service. Good cloud platform would ensure that this service is highly available and replicated as needed. Some of the important features of cloud load balancing are, number one: health check. Any load balancer you would want to create a health check because that's the only way a load balancer would be able to check if instance is healthy or not. So using health checks, cloud load balancing can cloud only to healthy instances. This would help you to recover from failures very quickly. The other important feature about cloud load balancing is

auto scaling. Based on the number of requests which are coming in, cloud load balancing would automatically scale. The other important feature is single Anycast IP. Cloud load balancing provides you with a single Anycast IP, and this IP can be used to receive traffic from multiple regions around the world, so you can serve global traffic using this specific IP address. In addition, cloud load balancing also supports internal load balancing. So, if you want load balancing for applications which are deployed within a specific network even that can be done using your cloud load balancing. The important reasons why we use cloud load balancing is because it enables high availability, even if the instances go down and come up, cloud load balancing ensures that the users do not get affected. Cloud load balancing will be distributing traffic only between the healthy instances. This also enables autoscaling because cloud load balancing by itself is autoscaling based on the number of requests, and also the backend services like the compute engines or the instance groups. All these would go down and come up over a period of time. By having a cloud load balancer we can actually create a very loosely coupled architecture. The other important thing is resiliency. Because we have health checks configured, the load balancer can detect if any of the instances is down, and it would distribute traffic only to the healthy instances. One other feature of the load balancer that ensures high availability is the fact that your load balancer can distribute load to instances in multiple zones and multiple regions. In this step, we got a high level overview of cloud load balancing. In the next step, let's understand some of the important terminology which are associated with load balancing. I'll see you in the next step.

Welcome back. Whenever we talk about virtual servers, load balancers, whenever we talk about communication between two different systems, we talk about protocols. We talk about HTTP, HTTPS, TCP, TLS, UDP. What are these protocols? Let's get a quick overview of all these protocols. If you already understand what is HTTP, HTTPS, TCP, TLS and UDP, you can safely skip this step. But if you don't really understand them, I would recommend you to spend some time watching this step. Whenever two systems talk to each other what would happen is the communication happens over multiple layers. The important layers are Application Layer, Transport Layer and the Network Layer. Why do we need these layers? Let's look at it right now. Computers basically use a number of protocols to communicate just like humans use languages to talk. So I can talk in English or Telugu, which is my mother tongue. Each of these languages have a proper structure and syntax. Similar to that, computers use protocols. So if you want two systems to talk to each other they need to understand the common language which is called a Protocol. So HTTP, HTTPS, TCP, TLS, UDP are different protocols at different layers. So there are multiple layers and multiple protocols. The Network Layer is responsible for transferring the bits and bytes. Whenever we talk about computers it's zeros and ones. Who handles the zeros and ones? That's the Network Layer. It's responsible for transferring the bits and bytes. Whenever we talk about two systems talking to each other, this communication happens over a network. There might be a number of systems in between these two systems which would help them to communicate with each other. If we are communicating over the internet, then the communication would happen over a number of routers. The next important layer is the Transport Layer. The Transport Layer is responsible to ensure that the bits and bytes, the zeros and ones are transferred properly. As we discussed, there are a number of systems which are in between, and if one of the systems in between, one of the routers in between goes down or something happens due to it you lose data, the transport layer is responsible for ensuring that the bits and bytes which this system sends are received properly by this system. The next important layer is Application Layer. The Application Layer is the top of the chain. The Application Layer is where we build most of our applications. We make REST API calls, we send emails, all of them work at the Application Layer. The thing is, each layer makes use of the layers beneath it. So even though we are using HTTP and HTTPs and they fall in Application Layer, when we use our Application Layer we are actually making use of the layers beneath it as well. So when we make a HTTP request, what it does is, it makes use of something called TCP in the Transport Layer and IP protocol in the Network Layer. So that's all beneath the hood and you would not need to really worry about it most of the times. Another interesting thing is the fact that not all applications would communicate at the Application Layer. If your application needs high performance, let's say, you are doing a streaming application or you are developing a gaming application, you want high performance, and in those kind of scenarios you might even skip one of the layers. Let's say you can skip the Application Layer and directly talk at the Transport Layer. So in summary, there are multiple layers which are involved when two systems talk to each other. The Network Layer is responsible for bits and bytes. The Transport Layer is responsible for ensuring that the bits and bytes are transferred properly. And the Application Layer is typically where we build most of our applications. Now, let's look at some of the protocols which are used at each of these layers. The Network Layer uses a protocol called IP, Internet Protocol. This is the protocol which is used to transfer bytes and bits. And the Network Protocol is very unreliable. Because when this sends some information, it goes over a network and the data might get lost over the network. And that's why we have the Transport Layer. In the Transport Layer, there are two important protocols. The first one is the Transmission Control Protocol. The Transmission Control Protocol gives high importance to reliability. If I'm sending 10 bytes of information, it would check that on the second system, the 10 bytes arrived as they are and in the right order. The next important protocol to understand is the TLS, Transport Layer Security protocol. It is very, very similar to TCP.

However, because data is going over the network you'd want to make sure that it is encrypted. So TLS is actually a secure form of TCP. So it's kind of TCP plus plus. The other important protocol is UDP, User Datagram Protocol. UDP is an alternative to TCP and it gives high importance to performance over reliability. If I'm sending a set of bytes over a network, and I'm okay with losing some of those bytes, there are applications where you would want to send hundred bytes over the network, and it's okay to receive only 98 of them on the other side. But for these applications, performance is very, very important. You would want to be able to receive these bytes immediately on the other side. A good example of this, is video streaming applications. In the video, even if a few bytes are misplaced that's okay. We don't really worry about it. As long as the majority of the picture appears good and it's streaming fast, most of the people would be okay with it. User Datagram Protocol is used in those scenarios. And the next layer is the Application Layer. On the Application Layer, we are very, very familiar with the HTTP protocol, Hypertext Transfer Protocol. It helps you to build web applications and things like this. This is a stateless request response cycle. And HTTPS is a secure version of HTTP. You don't want communication over a network which is unsecured. And HTTPS helps you to secure the HTTP communication by having certificates. It uses certificates which are installed on the servers to ensure that the communication between the two systems is secured. There are a lot of other protocols which are present at the application layer. Things like SMTP, which is used for emails, things like FTP, which is used for file transfer and a wide variety of them. In summary, most applications typically communicate at the application layer. So we are talking about HTTP, HTTPS, which is used by web apps, REST API. We have email servers. We have file transfers. All of them work at the Application Layer and they use their respective protocols. Applications using the Application Layer are actually making use of the layers beneath them. Typically, in the Transport Layer, they use TCP or TLS. Whenever we are sending a web application request response we would want to make sure that the bytes arrive safely and accurately on the other side. That's the reason why we use Transmission Control Protocol. HTTPS web applications would be using TLS, which is the encrypted Transmission Control Protocol. However, some of the applications which need high performance directly talk at the Transport Layer. The examples which we talked about are gaming applications or live video streaming applications, which can use the UDP protocol because for them hundred percent reliability is not important. For them, they can sacrifice reliability for the sake of performance. Now, the objective of this specific video was to help you understand the big picture. You don't really need to understand the bits and the bytes of all this stuff. If this gives you a big picture overview of how things work that's more than sufficient for you to understand whatever we would be talking about in the next few steps. Things like this are typically not discussed in all the courses but I thought, having a big picture overview of what happens underneath the hood, would help you to understand what we would be talking about in the next steps, very easily. I'll see you on the next step.

Welcome back. Are you excited to create your first load balancer? Let's get started. What we want to do is to create a load balancer and distribute the load between all the instances in the My Managed Instance Group. I would type in load balancing in here. So be careful, it's load balancing. And once you type in load balancing, you'd go to load balancing in here. Load balancing is part of network services and this is where you can create a load balancer. So a load balancer distributes incoming traffic among multiple VM instances. Let's go and say create load balancer and this would show the different types of load balancers that we can create. As you can see, there are HTTPS load balancers, TCP load balancers and UDP load balancers. UDP is layer for load balancing for those applications which are making use of UDP protocol. You can either create a internet facing or a internal load balancer. This depends on whether you have external users who are on the internet and using your load balancer or you just want your load balancer to be available just inside your specific network. One restriction on the UDP load balancing is that it can only be single region. You cannot create a multi-region UDP load balancer. The other layer for option is TCP load balancing. So this is layer for load balancing for applications that make use of TCP or SSL protocols. Typically, gaming applications need high performance. That's when you would go for layer for load balancing with TCP or SSL. Over here you have multiple load balancing options. Either you can go for TCP load balancer or SSL proxy load balancer or TCP proxy load balancer. And you can create either internet facing or internal load balancers as well as you have the option to deploy either in a single region or multiple region. Most of the times, we are creating HTTP or HTTPS applications, web applications, rest API. All these things make use of HTTPS load balancing. This is layer seven load balancing for HTTP and HTTPS applications. So you can either create a HTTP load balancer or a HTTPS load balancer. And similar to TCP load balancing, you can create either a internet facing or a internal load balancer and you also have the options to create it in either a single region or in multi regions. Let's go ahead and start configuring our load balancer. I'll say Start Configuration. So the first question that I would need to answer is do you want to load balance traffic from the internet to your VMs or only between VMs in your network? Let's take the default which is traffic from internet. The next thing we need to choose is which version of the HTTPS load balancer to use. Do you want to use the classic one or do you want to use the new version of the HTTPS load balancer with advanced traffic management? The new version is still in preview so what I would do is I would choose the classic HTTPS load balancer and I would say Continue. Inside the load balancer at a high level, there are

three important things that you need to configure. What is the backend? What are the host and path rules and what is different then? Backend is where we would want to redirect the incoming traffic to. In our case, our backend is nothing but a managed instance group. We'd want to send all the traffic to managed instance group. In the host and path rules, you would configure rules saying how you can redirect traffic to the backend and the front end configuration is where you would receive the traffic. The front end provides you with a URL for the load balancer so you would get traffic coming to the load balancer, that's basically the front end and based on the rules you configure, the traffic is redirected to the backend which is your instance group. Let's start with configuring the backend. The backend that we want to make use of is a managed instance group. So I would go ahead and select a managed instance group in here. So over here, I would choose create a backend service. You can see that you also have the option to create a backend bucket. A little later in the course, we'll be talking about cloud storage and how you can create buckets in cloud storage. You can use cloud storage buckets as a backend as well. For now let's focus on creating a backend service. When you choose to create a backend service, you can give it a name. The name I would want to give is My Instance Group Backend Service. So my instance group backend service. I'll choose the backend type is instance group. I'll take the defaults for protocol and timeout. If you scroll down, you can choose the instance group. So I can say My Managed Instance Group. This is the instance group which would be the backend and you can configure the port number. The port number we are making use of is 80, HDP port 80. The next thing you can choose is the balancing mode. Whether you'd want to distribute load among the instances based on the instance utilization or you can decide it based on the rate of requests. You can say I'd want a maximum backend utilization of 80% or I can say I'd want to handle a maximum of this many requests per second per instance. I would go with the default which is utilization and I'll say it done. Right now we have one backend configure. You can see that you can have multiple backends configured as well. If you're running a microservices architecture and you have one managed instance group for each of the microservices, then you can create multiple backends in here as well. Next, you can configure if you'd want to use a Cloud CDN. A Cloud CDN will help you to cache static content. Things like HTMLs, JavaScript files. These things are what is called static content and these files might not change often. So if you're load balancing that kind of content, then you can make use of Cloud CDN to cache the static content. Let's not really worry about it right now and we'll configure a health check. I'll reuse the health check which we have configured for our instance group, My VM Health Check. I'll take the defaults for the rest of the settings and I'll say Create. Once the backend service is created, I can go ahead and choose that and say Okay. so we have configured our backend. The next thing you can configure is host and path rules. The host and path rules decide how your traffic will be redirected to your backends. If you're running, if you are running a microservices architecture and let's say you are getting requests to the load balancer for multiple microservices, then you can add multiple host and paths rules and I can say if a request is coming to /microservice1, then I would send it to microservice1 backend. If the request is coming to let's say /microservice2, I can send it to the microservice2 backend and so on and so forth. In our example, we have a very, very simple requirement. All the requests which come to the load balancer, we would want to send it to this same backend which is My Instance Group Backend Service. So I'll choose that and leave the host and path empty. This is basically a simple host and path rule where we are redirecting all the requests to our instance group. If you have advanced needs like URL redirect or URL rewrite. In those kind of situations, you can also configure advanced host and path rules. Next thing you'd want to configure is the front end. Where do you want to receive the request at? The most important configuration which is present in here is the protocol which is HTTP and the port which is 80. These look good, so let's click Done and let's go over to review and finalize. So this is where you can see the configuration of the front end. You can see that we are making of HTTP and port 80. We have configured host and path rules so that any request which comes to the front end is directly sent over to the backend and the backend which we are making use of is an instance group with the name My Managed Instance Group. So everything looks good so let's go ahead and click Create. Now before you create a load balancer, you would need to give it a name. Let's go ahead and give it a name of My HTTP Load Balancer and go ahead and say Create. The creation of the load balancer would take a long time. Typically this takes somewhere between eight to 10 minutes. What I would recommend you to do is to take a break, grab a coffee and I'll see you on the other side.

Welcome back. In this step, let's review some of the important terminology that is associated with cloud load balancing. Let's consider this example. I have a number of instances which are created using a managed instance group, and I would want to route traffic between them using a cloud load balancer, and there are users who are sending the request to cloud load balancing. Considering this scenario, let's look at the terminology. Let's start with the backend. Backend is a group of endpoints that receive traffic from a cloud load balancer. For example, instance groups. So the backend is nothing but our managed instance group that we created, so this is the set of instances that are part of our managed instance group. If you're having microservices architectures then you can have multiple backends that are served by a single load balancer. You can create a backend for each of your microservices so you can create a microservice A backend service, microservice B backend service. The next important terminology is the front end. The front end is nothing but the address at which your cloud load balancer is available. How can user send

requests to the cloud load balancer? They need to specify a IP address, and they would need use a specific protocol and port to reach the cloud load balancer. So this IP address is the front end IP for your client requests. If you're using SSL, then a certificate is assigned to your cloud load balancer. In addition to the backend and the front end, you can also configure host and path rules. This is specific for HTTPS load balancing where you can actually define rules, redirecting the traffic to different backends. As we discussed earlier, you can have multiple backend services, for example, serving different microservices. And based on the path, for example, if the request is coming to in 28minutes.com/A versus in 28minutes/B. You can redirect it to different microservices. You can also redirect based on the host, so if the request is coming to A.28 minutes.com versus B.28minutes.com, you can redirect to different backend services. You can also go by different HTTP headers, like authorization headers or you can also go by the method, the specific HTTP method which is used to send the request. One more important concept that you need to understand about load balancing is SSL termination or SSL offloading. This can also be TLS termination or TLS offloading. If you're using layer seven then you're doing SSL termination or SSL offloading. If you're using layer four, and you're using security then you're doing TLS termination or TLS offloading. Let's assume a scenario where you are exposing an application on the internet. Then the client to the load balancer is going over internet. And for this traffic HTTPS is recommended. You want all the communication which is happening in here to be secure, and you can achieve that by assigning a SSL certificate to your cloud load balancer. Now, the traffic between load balancer and the VM instance, this is going through the Google internal network, and therefore HTTP is okay. While HTTPS is preferred, it's okay to use HTTP for this specific part of the journey, and that's the reason why SSL termination or offloading is important. So if you're using SSL termination, that's basically you are communicating on layer seven using HTTPS. Then decline to load balance of communication happens using HTTPS, and the load balancer to the VM instance communication happens using HTTP. So SSL or the secure part of communication is being terminated at the load balancer, and that's the reason why this is called SSL termination or SSL offloading. If you're using layer for communication this is called TLS termination or offloading, and between client and load balancer, TLS is used, and between the load balancer and the VM instance, we have unsecured communication, which is through TCP. The advantage of going for SSL termination is that you are reducing the load on the instances. Instances don't really need to worry about HTTPS or TLS. They can directly handle the requests. In this step, we understood some of the important concepts related to load balancing. I'll see you in the next step.
Welcome back. The creation of the load balancer took about 10 minutes and the end of it, I can see that my load balancer is ready. So if I go and look at the backends, I can see a backend present in here, my instance group backend service. And you can also see that there is a front end which is where the traffic is received from. And if I go to the load balancer and I click the load balancer, then I can see the port on which the load balancer is running. So let's pick up this port and go and execute the request against this. So you can see that as I execute the request, the load is balanced between the existing instances. Now if you have any problem with this, the things that you need to check are number one, give sufficient time for your load balancer to load up. The load balancer initialization would take about 10 to 15 minutes so give it a few minutes and then check afterwards as well. The thing you can check is this status of your backend. So you can see that for me, healthy is three by three. So all my instances are healthy. If you're having a problem, then make sure that you go to the VM instances and try the external IP of it and see that it's working. When I go into the load balancer I can see different details. The host and path details and the backend. And you can also see the health status of the backend services. You can also look at the monitoring. So if you go to the monitoring, you'll be able to see data around the load balancer. Right now, I don't see any data because I've just created the load balancer. If you give it a little bit of time, you should also be able to see the data around monitoring for this specific load balancer. Now let's quickly review the hierarchy. So we have created a load balancer to load balance between a set of instances. How are these instances created? Let's go back to our instance template. So we created a instance template defining how a specific VM instance should look like. And that VM instance is my instance template with custom image. And using this instance template, we have created a instance group. What is the instance group that we have created? The instance group that we have created is my managed instance group. And this instance group uses this specific template. And in the instance group, we are configuring how many instances that we would want to be running all the time. And after that, we created a load balancer which is tied to this specific instance group. How did we tie the load balancer to this specific instance group? We created a backend. We created a backend service, which is mapped to this specific instance group. And the load balancer, whenever it receives traffic on the front end, it automatically distributes it to the back end. And that's how you can see that when I am actually refreshing this multiple times, you would see that the request is being sent to multiple VM instances. It is very, very important to understand the entire hierarchy as far as the exam is concerned. So what I recommend you to do is to try and spend a little bit more time thinking about what we have set up in here.
Welcome back. In this step let's understand how to choose which load balancer to use. As you can see, there are different criteria that can be used to choose a specific type of load balancer. Whether your traffic is coming from internet or not. Do you have external traffic or not, and what kind of traffic are you getting? HTP or TCP or UDP.

You can see that the choice for external traffic which is coming in from internet, is pretty complex. The choice for internal is pretty simple so let's start with internal. So if you're traffic that is internal to a network then the important question is what kind of traffic? Is it TCP or UDP traffic or is it HTTP or HTTPS traffic? In the case of TCP or UDP traffic you can make use of internal TCP UDP load balancing. In the case of HTP or HTTPS traffic you can make use of internal HTTPS load balancing. So that's the simple part. Now, let's get to the complex part. If you have traffic coming from internet and you'd want to load balance it then the first question is, is it HTTPS or TCP or UDP or ICMP? You don't really need to worry about ICMP because it's something which is used in case of things like router. So let's focus on HTTP, TCP, and UDP. If your traffic is HTTP or HTTPS it is very important to remember that there is a classic HTTPS load balancer which is kind of the old version of the HTTPS load balancer and there is also a new version of the HTTPS load balancer with advanced traffic management. If you go to the new version of the HTTPS load balancer you can see that you can either go for a global one or a regional one as well and that is a choice which is represented in here as well. So if you're HTTP or HTTPS traffic is specific to a specific region, then the recommended option is to go for the new version of the HTTPS load balancer and choose regional HTTPS load balancer. So that's what is shown in here. If you have global traffic, then you can either choose between the classic HTTPS load balancer or you can choose the global HTTPS load balancer new version. One thing that you need to have in mind is the new version of the HTTPS load balancer provides you with advanced traffic management. If you want to modify your request or response header, or if you'd want to retry failed requests, or if you'd want to implement things like traffic splitting or traffic metering all those advanced features can be implemented within new version of the load balancer from HTTP HTTPS. Let's move to TCP traffic. In the case of TCP traffic, if you would need SSL offloading, then you would need to go for SSL proxy. The next question you need to ask is if it's global load balancing or ipv6 load balancing. If that's the case, then you would need to go for TCP proxy and the next question you'd think about is if you'd want to preserve client IPs. Do you want your backend applications to see the request as is as it is coming in from the users? If the answer to preserve client IPs is yes, then you would go for network load balancing from TCP, Let's move to UDP. If you have external UDP traffic the only option you have is to go for external network load balancing. The same is the case for ESP or ICMP traffic. Understanding how to make this choice is very very important as far as the exam is concerned. So it's very very important to remember this specific chart. So if you have internal TCP or UDP traffic you'd go for internal TCP UDP load balancing. If you have internal HTTP or HTTPS traffic you'd go for internal HTTPS load balancing. In the case of external, the choice is little complex. If you have HTTP or HTTPS traffic then the first question is do you want to create a regional load balancer? If you want to create a regional load balancer you'd want to make use of the new version of the HTTPS load balancer and choose the regional option. If you want a global load balancer in that kind of situation, you can either choose between the classic one or you can go for the global HTTPS load balancer option which is presenting here in the case of TCP traffic. If you want SSL offloading then you'd need to go for SSL proxy. If you don't want SSL offloading and you want a global load balancer or a ipv6 addresses, then you'd go for TCP proxy. If you don't want to preserve client IPs you'd want to go for external network load balancing external network workload balancing is also recommended for UDP and ICMP traffic. I'm sure you're having an interesting time and I'll see you in the next.

Welcome back. In the previous steps we played around with the Cloud load balancer. In this step, let's look at some of the important features of the different load balancers. External HTTP, this is used for global external HTTP or HTTPS traffic and this is a proxy load balancer. At a high level difference between proxy and pass-through is that proxy load balancers get the request from a client and they transform it or they make changes in it and they send a different request to the backend. In the case of pass-through, whatever request comes from the client directly is sent out to the backend. So the client will be able to see all the details of the request, which is sent by the client as is. So the backend would be able to see the IP address of the client, for example. So the external HTTP is a proxy load balancer. The ports on which it can load balance is either 80 or 8080 or 443. HTTP on 80 and 8080, HTTPS on 443. The internal HTTPS is for regional internal traffic. Again, the type of the traffic, which itself is HTTP or HTTPS and this is also proxy and the port remain the same. SSL proxy is for layer four global external TCP traffic with SSL offloading. Again, this is a proxy and there are a big list of ports that you can listen on. You'd go for TCP proxy when you have global external TCP traffic without SSL offloading. And you'd go for external network TCP/UDP load balancer, when you have regional external traffic with TCP or UDP. Important difference of external network TCP/UDP load balancer compared to all the other load balancers is the fact that this is pass-through. The backend will be able to see the client request as is, so it can see the client IP address. The other important difference is the network load balancer can listen on any of the ports. The last type of load balancer is the internal TCP/UDP load balancer. The only difference between the internal TCP/UDP and the external network TCP/UDP load balancer is the fact that the internal TCP/UDP is a internal. It's only for traffic within a specific network. The idea behind this step was to quickly review some of the important differences between the different load balancers. I'm sure you're having an interesting time and I'll see on the next step.

Welcome back. Now that we played with Cloud Load Balancing, let's look at a very important use case with respect to load balancing. You have a microservice and for that microservice you have multiple managed instance groups which are created in multiple regions. So you have a managed instance group in region A for the microservice and also a managed instance group in region B for that microservice. The great thing is you can use HTTPS Cloud Load Balancing to load balance between managed instance groups which are deployed in multiple regions. Let's see a few more details in the specific step. A regional MIG can distribute instances in different zones of a single region. If I create a regional MIG like this, the instances of microservice A would be distributed across multiple zones of this specific region. So this would ensure high availability. If you want higher availability, what we can do is we can create multiple regional MIGs in different regions within the same project. So inside the same project, for this microservice A I would create a managed instance group in region A and also in region B. With this setup, you would get even higher availability. Even if the entire region goes down you can serve the users from the other region. You can set up Cloud Load Balancing to distribute load between these MIGs which are present in multiple regions. An amazing thing is as far as the users are concerned, they would need to use this same external IP address. So you have a single external IP address and Cloud Load Balancing can redirect requests to the nearest region for the user. If the user is in region A, then the request would be sent to region A. If the user is sending the request from region B, then the request would be sent to the managed instance group in region B. This would ensure that your users get low latency. The Cloud Load Balancing would ensure that the request is served from the instance group which is nearest to the user. The other thing Cloud Load Balancing also enables is that Cloud Load Balancing only sends traffic to healthy instances. You can configure a health check on the load balancing and if the health check fails, the instances are restarted. One of the important things that you need to remember is to ensure that the health check which is executed by Cloud Load Balancing can reach the instances which are part of the instance group. How can you do that? You can ensure that there are firewall rules configured that access from Cloud Load Balancing for that specific instance. If you don't allow health check access from load balancing to a specific instance, what would happen? The health check would fail and the load balancing would think that the instance is down and it would keep restarting it and you don't want that. Another possibility is that all the back ends within a specific region are unhealthy. Let's say for some reason, all the back end to all the instances which are present in region A are down. In that kind of a scenario, Cloud Load Balancing would automatically distribute load only to the instances in region B. So the traffic in those kind of situations is distributed only to the healthy back ends in other regions. In this step, we talked about the fact that you can use load balancing to load balance between manage instance groups which are present in multiple regions. Let's discuss a little bit more about this in the next steps.

Welcome back. In the earlier step, we talked about the fact that HTTPS load balancing can load balance between managed instance groups which are deployed to multiple different regions. When we talk about a microservice architecture you have a number of small microservices instead of large monolith. And each of these microservices can be deployed to multiple different regions, and each of these microservices can also have multiple versions. So there are a lot of scenarios that are possible with deploying your microservices. In this step, let's look at the important load balancing concepts and how you can use them to implement different scenarios related to microservices. Earlier in the course, we created a HTTPS load balancer. We said create load balancer on the load balancing screen of the console, and we said let's configure a HTTPS load balancing and I would want to load balance traffic from internet. So let's say continue. And this is where you saw that there are three important things that you need to configure as part of your load balance at configuration. The first one is backend configuration. The second one is host and pas rules. The third one is front end configuration. The front end configuration is very important to your users. This gives you a IP adders, a protocol, and a port and this is what is used by your users to reach your load balancer. Once a user sends a request to the load balancer the request should be routed two different microservices. They can be multiple versions and they could be multiple regions as well. How does the load balancer decide which instance should handle the request? That's where hosting pas rules and the backend configuration come into picture. As part of your backend configuration there are two important things that we would configure. One is called a backend service and the other one is a backend. As part of your host and path rules, what we are configuring is how your traffic will be redirected. And this configuration is done by using a URL map. Let's consider a very simple example. You have two microservices, and for each of these two microservices, you have two managed instance groups in two different regions. In this kind of scenario, what we would do is we would create a backend service for each of the microservices. So backend service A represents microservice A. Backend service B represents microservice B. And what we would do is we would create a URL map with the following routes: The first route would say, If a request comes to slash service A, it needs to go to microservice A. If the request comes to slash service B, it needs to be sent to the backend service of microservice B. And behind these backend services are the backends. We talk about the fact that microservice A has two managed instance groups in two different regions. Each managed instance group is represented by your backend so a backend is nothing but a managed instance group in a specific region. Because the microservice A has two manage instance groups we have configured two backends

for it. Similar to that, microservice B also has two backends. These represent the manage instance group of microservice B in two different regions. So a backend service is nothing but a group of backends that represent a specific microservice for example. The backend service can also be a cloud storage bucket. A little later we'll be looking at cloud storage which is object storage service in Google Cloud. If you have some static content in Google Cloud you can also route to it from cloud load balancing. A backend represents a specific set of instances. For example, a managed instance group and the URL maps help you to route requests to backend services or the backend buckets. To help you understand this better, let's look at three different use cases and let's see how you would configure backend service, backend, and URL maps for those scenarios. Let's start with a very simple scenario. Let's say we want to create a multi-regional microservice. I want a single microservice deployed to multiple regions. If you're using Compute Engine, the easiest way to set this up would be to create instance groups in each of these regions. So we can create instance groups in each of the regions you'd want to deploy your microservice to and you can create a backend service to represent the microservice. So we have just one microservice so we'll create one backend service for it and you can create multiple backends, one each for each of the managed instance groups of the microservice and you can configure a simple URL map. All that the URL map needs to do is when the request comes to service A, it needs to map to the backend service of this specific microservice. As we discussed earlier this provides you with global routing. The cloud load balancing would automatically route user to the nearest instance or the nearest regional MIG. One important thing that you need to remember about using global routing is when you are using global routing, you need to use the premium tier of networking. There are two networking tiers which are provided by Google Cloud - Premium and Standard. Premium provides you with optimal performance. Premium is the tier which Google users to run all its importance services like YouTube, Gmail, et cetera. If you want to reduce your cost, you can go for standard tier, but in standard tier you need to remember that the forwarding rule, whenever we configure a cloud load balancing, the first thing that we would need to configure is a front end. And the front end has a forwarding rule, and the IP address of the front end rule in standard tier is regional. So you can only have backends for that specific backend service in the same region. So if you're using standard tier, remember that all your backends should be in the same region, so you cannot do global routing like this. If you want to do global routing you need to make use of the premium tier. Now, let's complicate this scenario a little bit more and bring in another microservice. So if you have another microservice let's say microservice A and microservice B how would your backend surveys backends and URL maps look? It should be simple, right? You have backend services one backend service for each microservice. Now, when it comes to backends you can create a backend for each managed instance group of that microservice in that specific region. So each microservice can have multiple backend images in different regions. However in the example in the picture that we are showing in here we are just seeing one backend per microservice. You can have multiple backends as well. And the URL maps, as we discussed earlier, would be simple. Service A, you would redirect it to microservice a backend service. Service B, you redirected to microservice B backend service. The amazing thing about this architecture that we have set up is that you can extend it easily even when you have multiple versions of multiple microservices. Let's consider an example where you'd want to have two versions of microservice A. You just have one version of microservice B but you'd want to serve two versions of microservice A and you'd want both these versions to be live at the same time. You have some users going to version two of microservice A. You have some users going to version one of microservice A. As you can see in here implementing this is very, very simple. You'd create backend services for each version of the microservice that you would want to deploy. So over here we create a backend service for microservice AV1, microservice AV2 and we just have one version of the microservice B. So it has one backend service, so a backend service for each microservice version, and you can create backends for whatever backends you would want to have for that version of the microservice. So if I would want three backends for version one and two backends for version two you can have those many backends present. However, in the example that we are seeing in here we just have one backend per microservice solution and the URL maps configuration should also be straight forward slash service AV1. It goes to V1 backend service. V2 goes to V2 backend service. Service B goes to microservice B backend service. As you can see, HTTP load balancing provides you with a very flexible architecture. You can route global requests, you can route requests coming from users around the world, across multiple versions of multiple microservices, and you can serve the users from the nearest backend to them. The idea behind this step is to show you the flexibility in the architecture that HTTPS load balancing enables. I'm sure you're having a wonderful time and I'll see you in the next step.

05 - Google Cloud Compute Engine & Load Balancing for Architects

Instructor: Welcome back. Welcome to this section on Compute Engine and Load Balancing for architects. There are a lot of different things that architects need to worry about. Availability, scalability, resilience, cost efficiency, performance, security, and that's what we would be looking at in the specific section. In the initial steps of this section, we'll be focusing on availability and scalability. We'll understand how you can build highly available and highly scalable solutions for Compute Engine and Load Balancing. After that, we would switch our attention to security, performance, and resilience. At the end of this section we'll be looking at sustained used discounts, committed used discounts, and preemptible VMs. These are some of the important options to build cost effective Virtual Machines using Google Cloud Compute Engine. Are you ready to get started with looking at the different architectural aspects for Compute Engine and Load Balancing? I'll see you in the next step.

Welcome back. Welcome to this new section Compute Engine and Load Balancing for architects. We have looked at a lot of details related to compute engine, manage instance groups, and load balancing in the last few sections. However, our goal with this course is to help you become a real world architect. We want you to be able to architect great solutions in Google Cloud and obviously we want to help you do very well at the exam as well. And that's the reason why we added in sections specifically talking about architecture. Whenever we talk about architecture it is not sufficient to just get things working. We want a lot more. You want to be able to build resiliency. You want to be able to increase availability. You want to be able to increase scalability, improve performance, security, and reduce the cost at the same time. And there are a lot of other things that you would want to be able to do as well. If you're new to architecture, a lot of these terms might be new to you. The idea behind this architecture sections is to introduce these terms and also how you can make use of the different Google Cloud services to architect really good solutions. And that is one of the major differences between other certifications and the professional cloud architect certification. Let's take associate cloud engineer as an example. Associate cloud engineer is focused on tasks that cloud engineer perform on, in day to day job. However, when we talk about the professional cloud architect certification it is all about designing solutions in the cloud. And to be able to design solutions in the cloud you need to understand business requirements, you need to understand technical requirements, you need to understand nonfunctional requirements. And at the end, you'd want to be able to design solutions that meet all your needs. In the background, the services that we make use of the computer services the networking services, the database, and the storage solutions that you make use of are the same. However, your perspective when you are doing the professional cloud architect certification should be a little different. For this certification, you need to know these services and also you need to be able to build highly resilient, highly available, scalable, secure performance solutions that have lowest cost using these services. And for a big nerd to architecture, or even for somebody who's little bit of experience with architecture, all these might sound a little complex. Don't worry about it. It's my responsibility to help you understand all these terms as we go further, as well as help you build solutions that are really, really good. Are you excited to get started?

Welcome back. In this step let's get started with availability. What is availability? It's a very, very simple measure. Are the applications available when the users need them? It is the percentage of time and application provides the operations that are expected out of it. A good example of availability number that you would see often is 99.99%. It's also called four nine availability. If you see architects talk about four nine availability or five nine availability it actually converts to a number. Four nines availability is 99.99% Five nines availability is 99.999%. To get a different perspective on availability let's look at the availability table. We'll look at availability. What is the downtime allowed in a month if you want to adhere to that specific availability requirement and let's look at a few more details. 9.95% availability means a downtime of 22 minutes in a month. As you can see, even meeting 99.95% availability might be tough for some applications. Let's say you are doing five releases a month and each release would need five minutes of downtime. You need five into 5, 25 minutes of downtime, and then you not be able to meet the availability requirement. As you can see building highly available applications is not easy. If you increase the availability needs to 99.99% or four nines the downtime allowed reduces to four and half minutes so in a month your application can only be down for four and a half minutes or lower if you want to meet this availability requirement. Typically, most online apps aim for 99.99% availability. If you want higher availability you can try going for 99.999% availability which means 26 seconds of downtime for your app in a month and achieving five nine's availability is really, really really tough. It is very, very important when you talk about availability to remember that your application is not just your front end it's your front end, it's your APIs, it's your database. All these combined together, it's your application and whenever we talk about an application being available all the things that constitute the application should be available. In this step we talked about availability. Availability is nothing but a percentage of time an application provides the operations that that are expected out of it. We talked about the fact that most online apps aim for four nine or 99.99% availability. Let's talk a little bit more about availability in the next step.

Welcome back. In this step, let's look at high availability for compute engine and load balancing. How do you get

high availability for compute engine and load balancing? Let's look at an example architecture right now. You can create multiple regional instance groups for each microservice. As we already know a regional instance group distributes instances across a single region. And for even higher availability, we can distribute instances across multiple regions. And the way we can do that is by creating multiple instance groups in different regions. And you can distribute load between them using a global HTTPS load balancing implementation. One of the important things with respect to high availability to ensure that you can detect if a instance is down. If a specific instance is down, you'd want to find it early and you'd want to replace that instance. And how do you do that? The way you can do that is by configuring health checks, both at the instance group level and the load balancing level. The other important setting when it comes to compute engine virtual machines is to enable live migration. If you enable live migration, if there is a software or hardware update that needs to be performed by the compute engine, the VM instance would be live migrated to a different host, and that would ensure that you don't have any downtime. Now, what are the advantages of having such architecture? The instances are distributed across regions so we have instances distributed across multiple regions. That means you can survive a zonal failure as well as a regional failure. Even if one of the zones fails, it's not a problem you can serve users from the other regions and zones. Even if the entire region fails, you can serve users from the other region. The other important thing to remember is cloud load balancing is a managed service. It's a service managed by Google Cloud, and you don't really need to worry about availability for managed services. Google Cloud would take care and ensure that cloud load balancing is highly available and also it can auto scale based on the number of requests. Configuring health checks ensures auto healing, even if one of the instances is down, you can detect it and you can replace it with a healthy instance. In this step, we looked at a high availability architecture for compute engine and load balancing. I'll see you in the next step.

Welcome back. In this step, let's talk about, what is scalability? Let's take an example. A system is handling 10,000 transactions per second and the load is expected to increase 10 times in the next month, the load is expected to go to 10,000 transactions per second. Let's say I have five instances to support the system right now. A system is said to be scalable if I can increase the number of instances to 50 and handle the expected load. So the questions that we would typically ask when it comes to scalability are, can we handle a growth in users, traffic, or data size, increasing number of users or increasing amount of data that you would want to handle? Can we increase that without any drop in performance? And another important question is does the ability to serve more growth increase proportionally with the resources? If I'm able to do 1,000 transactions per second with one resource, can I increase the number of resources to 10 and handle 10 times the load? Does the ability to serve growth, whether it's in number of users or in terms of data increase proportionally with the amount of resources that you are making available for your system? Scalability is the ability of your system to adapt to changes in demand. Whenever we talk about online applications the number of users on the application keeps changing. If all of a sudden your application becomes really popular the number of users using your application increases drastically. Is your application able to adapt to the changes in demand? That's what scalability is all about. When we talk about scalability, there are multiple options. The first option is to deploy to a bigger instance with bigger CPU and more memory. Let's say I'm using a virtual machine with X amount of memory, I can increase the amount of memory to three x. That's one option. I can use a bigger instance. The other option that I have is to increase the number of application instances and set up a load balancer. So instead of having one VM, I can have 10 VMs and I can set up a load balancer to load balance between them. And there are a lot of other options. The first option where we increase the size of instances is called vertical scaling. Deploying an application or a database to a bigger instance a larger hard drive, a faster CPU, more RAM, CPU, I/O, or networking capabilities. This is what is called vertical scaling. Instead of using a small instance, you are going for a larger instance. However, it's very important to remember that there are limits to vertical scaling. There is a cap on the maximum amount of CPU an instance can have. There is a cap on the maximum amount of CPU that an instance can have. So if you use vertical scaling you can go only up to a certain limit. Beyond that, you would need to go for horizontal scaling. What is horizontal scaling? Horizontal scaling is all about deploying multiple instances of your application or database. So instead of having one instance of the application you'll have multiple instance of the application. Instead of having one server serving your database you can distribute the data across multiple different servers. Typically, horizontal scaling is preferred to vertical scaling. Why? Because vertical scaling has limits. The second thing is vertical scaling can also be expensive. Typically, you would see that the most powerful machines are very expensive. In most of these circumstances, buying two or three low end machines is cheaper than buying one very powerful machine and that's the reason why vertical scaling can be really, really expensive. The other important thing is there are additional advantages to horizontal scaling. Horizontal scaling also increases availability of your applications. If I'm increasing the size of the instance, in either case, I just have one instance. If that instance goes down, the application goes down. However, when I'm doing horizontal scaling, instead of having one instance I have 10 instances or 15 instances or 20 instances. Even if one of the instances goes down I can serve the application using the other instances. So horizontal scaling increases availability of your applications. However, horizontal scaling does not come free, it needs additional infrastructure. A good example is load balancers. In the earlier examples, we

saw that when we create multiple virtual machines you would need a load balancer to load balance between them. Scaling is all about the ability of your system to adapt to changes in demand. We saw that you can respond to increases in demand in two different ways. Vertical scaling is all about increasing the size of the instance. Horizontal scaling is all about increasing the number of instances. Let's talk a lot more about it in the subsequent steps.
Welcome back. In this step, let's look at vertical scaling and horizontal scaling from the perspective of G CEVMs Compute Engine virtual machines. Earlier we talked about the different machine types that are present in Compute Engine. We talked about the fact that there are multiple machine families and under each of the machine families there are multiple machine types. Let's consider the E2 machine family. Under E2 machine family you have different machine types e2-standard-2, standard-4, standard-6, standard-16, 32. And each of these have different amount of hardware that is associated with them. E2-standard-2 gives you two vCPUs, 8 GB of memory and can go up to a maximum egress bandwidth of 4 Gbps. And as you can see, as you go further down the list the amount of hardware that is present with that specific machine type increases. So if you're doing vertical scaling for your Compute Engine virtual machine, what you would be doing is you would increase the VM machine size. So you would go from e2-standard-2 to e2-standard-4 or e2-standard-8 or e2-standard-16. You could also go from one machine family to another machine family, or you can add a GPU or a TPU. All these are examples of vertical scaling too. As far as Compute Engine is concerned one important thing that you need to remember is before you do a change in machine type of a virtual machine instance, you need to stop it. You need to first stop the instance, change the machine type, and then you can bring the instance up. Now let's shift our attention to horizontal scaling. We already looked at examples of horizontal scaling setups for Compute Engine VMs. What we would do is we would create different instance groups for the same application in different regions and we would use cloud load balancing to load balance between them. So we would distribute VM instances either in a single zone, so if you create a zonal instance group then you can distribute instances in a single zone. Or you can create a regional instance group to distribute instances in multiple zones in a single region. Or you can create multiple managed instance groups to distribute instances in multiple zones across multiple regions. You can also configure auto-scaling for these instances. As the load increases, you can increase the number of instances by configuring auto-scaling you can auto scale based on CPU or any other metrics and we can automatically distribute load among the healthy instances by using cloud load balancing. In the step we look at how you can actually do vertical scaling and horizontal scaling for your Compute Engine virtual machines. I'm sure you're having an interesting time and I'll see you the next step.
Welcome back. In this step, let's look at another important feature of a compute engine, which is Live Migration. How do you keep your VM instances running when a host system needs to be updated? Let's say GCP wants to perform a software or a hardware upgrade on your specific virtual machine. You don't want the application to go down when the software or the hardware upgrade is being performed. To ensure that, compute engine provides a feature called Live Migration. What happens with Live Migration is that your running instance is automatically migrated to another host in this same zone. So if the software and hardware update is happening on a specific host, all the running instances on that specific host will be migrated to another host which is present in the same zone. And this does not change any attributes or the properties of the VM. Live Migration is also supported for instances even having local SSDs. However, it is important to remember that Live Migration is not supported for preemptible instances and instances to which you have added graphic processing units. Now, how do you configure Live Migration? How do you configure if you'd want to use Live Migration or not for your specific instance? The place you would want to configure this is availability policy. This is very, very important to remember as far as the examination goes. Availability policy is where you would configure the Live Migration configuration. And as part of availability policy, you can configure two important things. One is on host maintenance. What should happen during periodic infrastructure maintenance. The default is migrate, which is to migrate VM instances to another host, to other hardware. The other thing you can also do is terminate. If you don't want your instance to be migrated, then you can choose on host maintenance as terminate. Then the VM instance would be automatically stopped at the time of a migration. The other option you can also configure is automatic restart. In the availability policy, you can also configure if your VM instance has to be automatically restarted if they are terminated due to any non-user initiated reasons. So if you're not terminating the instance and due to a maintenance event or due to a hardware failure, the instance is terminated. In those kind of situations, you can also configure that the instance should be automatically restarted. This configuration also can be made in the availability policy. Let's see where the configuration is in the console right now. Now this is a screen from which you can create a VM instance. You can go in here. We already looked at availability policy configuration when we were talking about preemptible instances. It's the same place we need to go right now as well. So I would scroll down and go to management security discs. And if you actually scroll down a little and go towards availability policy, this is where you can see the configuration of on host maintenance and automatic restart. This is just below the preemptible configuration. So on host maintenance. When compute engine performs periodic infrastructure maintenance, it can migrate your VM instance to other hardware without downtime. And you can see that the default option which is recommended is migrate VM instance.

However, if you want, you can actually choose to terminate the VM instance as well. And in addition, you can also configure automatic restart. Compute engine can automatically restart VM instances if they're terminated for non-user initiated reasons. Maintenance event, hardware failure, software failure, or other such reasons. And you can leave the automatic restart at on, which is the recommended option and the default option as well. In this step, we looked at a important option to have high availability for your virtual machines. You can set on host maintenance to migrate and you can enable automatic restart. I'm sure having wonderful time and I'll see you in the next step.
-: (narrator) Welcome back. In this step, let's look at a very important computer engine feature as far as performance is concerned, which is called a GPU, a graphic processing unit. How do you accelerate math intensive and graphic intensive workloads? If you have graphic intensive or math intensive workloads which you are running for AI or ML. In those kind of scenarios, you can add a GPU, a graphic processing unit to your virtual machine. This gives you high performance for math intensive and graphics intensive workloads. However, there is a higher cost attached with it, and very important to remember is when you add a GPU make sure that you have this software that is needed to make use of the GPU installed on your virtual machine. So, make sure that you use images with GPU libraries which are pretty installed. If you don't do that, your GPU will not be efficiently used. There are a few restrictions when it comes to GPU as well. It is not supported on all machine types. For example, it is not supported on shared core on memory optimized machine types. Another important restriction we already talked about is that you cannot do live migration for instances with GPU attached. Your on host maintenance value should be terminate VM instance only. Typically, recommended availability policy for GPU is to have automatic restart set to ON. This will ensure that even if there is a hardware maintenance done the instance would be automatically restarted. Now, where can you configure a GPU? Now, we are on the screen where we can actually create a VM instance, and over here there are two ways you can actually create GPU machines. One is you can choose GPU machine family. So, you can go to the machine family, you can choose GPU, and there are specific machine families, which are GPU specific machine families. So, you can go and choose them. The other option is you can actually choose any other machine family and you can add GPUs to them. The GPUs are not available with all machine families, so if you actually select E two, which is default which is suggested for me, and you expand CPU and GPU you would see that the ADD GPU button is disabled. However, if I go and choose N one, for example, and scroll further down you'd see that I can add GPUs to it. So, you need to choose the right machine family and for those you can explicitly add GPU's as well. In this step, we talked about a very important computer engine feature called GPUs. GPUs are very, very important if you'd want to accelerate math intensive and graphic intensive workloads. There are two ways you can use GPUs with your virtual machines. One is opt for a GPU family. The other is to opt a normal family and add a GPU. One important thing to ensure is that you're using the right image. You need to use an image with the GPU libraries installed. Otherwise, your GPU will not be efficiently used. I'm sure you're having interesting time and I'll see you in the next step.
Instructor: Welcome back. Having looked at compute engine and load balancing from the perspective of scalability and availability. Let's now shift to other perspectives. Let's start with security. How can you ensure that your implementations are really, really secure? Number one is you can use firewall rules to restrict traffic. You can control what traffic is coming in and going out of your virtual machines. You can say, I would want to only allow HTTP or HTTPS traffic into your virtual machine and you can say, I only want to be able to call this website from your virtual machine. I would want to only allow this protocol from my virtual machine. You can do this by configuring something called firewall rules. We'll talk about firewall rules a lot more when we get to the networking sections of this course. But for now, the important thing to remember is you can use firewall rules to rustic traffic that is coming in and going out. Coming in traffic is called ingress and going out traffic is called egress. So you can control ingress and egress from your virtual machines using firewall rules. The second important recommendation is use internal IP addresses as much as possible. Avoid external IP addresses. Internal IP addresses will ensure that nobody can directly access your virtual machines from outside, so this will give you additional security. The third important thing is use, sole-tenant node when you have regulatory needs. If you have compliance needs where all the virtual machines that are running on a specific host needs to belong to you in those kind of situations, you can go for sole-tenant nodes. The last, but the most important recommendation as for a securities concern is to create a hardened custom image to launch your virtual machine. Whenever we create a virtual machine, we choose the image from which we would want to launch up that virtual machine. Make sure that that specific image is customized and hardened to meet the security needs of your enterprise. Let's now shift our attention to performance. The most important decision that you make with respect to performance of your virtual machine is choosing the right machine family. So choose the right machine family for your workload depending on the type of workload you are running. If you're running a cache, then you need a lot of memory. If you're workloads, which do a lot of computations then you need a lot of CPU. Based on that, you need to choose the right machine family. Also, remember that you can use GPUs and TPUs to increase performance further. So if you have machine learning or data processing workloads, you can add GPUs to your virtual machine. If you're performing massive metrics operations in your machine learning

workloads, you can also add in TPUs. The last performance recommendation is to ensure that you create a hardened custom image. Rather than installing software at launch of a virtual machine, it's preferred to create a hardened custom image with the software pre-installed so that the startup of virtual machine is faster. In this step, we looked at some of the important things that you can do as architects to improve the security and performance of your virtual machines. I'm sure.
Welcome back. In this step, let's understand what is resiliency, and how you can build resiliency into your implementations. What is resiliency? Ability of your system to provide acceptable behavior, even when one or more parts of your system fail. Let's take this example. This is a good example of a resilient solution. We have multiple instance groups created for the microservice in different regions, and you're using Cloud Load Balancing to load balance. Why is it resilient? Because even if one of the instances in a specific microservice instance group fail, you'd be able to easily recover from it. Even if the entire instance group in region A goes down, we can serve our users from the instance group in region B. So even if there are failures in parts of the system, even if there are failures in one or more parts of the system, we can provide acceptable behavior back to the user. It is very, very important for you as an architect, to focus on building resilient architectures. A good resilient architecture when it comes to virtual machine, is what we have been looking at until now. Virtual machines in a managed instance group behind global load balancing. One of the most important things when it comes to resiliency is to ensure that your team has the right amount of information available. You can look at the metrics behind your virtual machines behind Cloud Load Balancing using Cloud Monitoring. So make sure that you use Cloud Monitoring for monitoring. The other important thing that you would need to quickly react to problems, is to have lot of logs. So ensure that on each of the VMs, you have logging agents installed so that these logs can be sent out to the centralized logging solution in Google Cloud, which is Cloud Logging. We'll discuss a lot more about Cloud Monitoring and Cloud Logging a little later in the section on operations. The attitude that you need to have to build resilient solutions is to be prepared for the unexpected. Be prepared for things to change. Some of the things that you can do is to enable live migration on your VM. Enable automatic restart when the feature is available for you. Make sure that you have the right health checks configured, so that even if something goes wrong, the instance is restarted. Make sure that you have an up-to-date hardened image that is copied over to multiple regions, so that even if one of the regions is not available, you can use the image from the other region. Later in the course, we'll also be talking about system reliability engineering, and over there, we would be talking about resiliency in much further depth. I'm sure you're having a wonderful time, and I'll see you in the next step.
-: Welcome back, until now we have been playing with virtual machines. Starting this step, let's look at a few important things that you need to understand about keeping your costs low. Whenever you are making use of the cloud, you'd want to keep your costs as low as possible. What are the things that you can do to keep the, to keep the cost of your virtual machines, low? The first thing that we'll be talking about is Sustained Use Discounts. What is a sustained use discount? It's an automatic discount, which is applied, for running VM instances, for significant portion of the billing month. For example, if you're using N1 or N2 machine types for more than 25% of the time period of the month, then you'd get about 20 to 50% discount on every incremental minute. So beyond 25%, whatever time period you are running for you'd get a discount on it. And the discount increases with the usage. If you are using the VM for 80% of the month, the amount of discount you would get would be higher than the amount you would get if you're running it for 40%. And the great thing about the sustained use discounts, is that you don't really need to do anything. There is nothing you need to do specifically for this when you're creating or running your virtual machines. This is automatically applied by the platform itself. However, there are a few important restrictions that you need to remember. A little later in the course, we'll be also talking about Google Kubernetes engine, which is a managed Kubernetes service, which is provided by Google Cloud platform. Whenever you're creating instances using the Google compute engine, that's basically what we are using in the course until now. Or when you create instances using Google Kubernetes engine, the sustained use discounts are automatically applied. Remember that this does not apply on certain machine types. For example, E2 and A2, the sustained use discounts do not apply for them. And also remember, that if you're using App Engine flexible or Data Flow, to create the VMs, sustained use discounts do not apply. We'll talk about App Engine and Data Flow as we go further in this course. The important thing to remember about sustained use discount is the fact that these are automatically applied, when you use instances which are created by Google Kubernetes engine and compute engine for a significant part of a specific month. Also, remember that there are a few restrictions. I'm sure you're having an interesting time and I'll see you in the next step.
-: Welcome back! In this step, let's look at another type of discounts which are applied on your virtual machines. How would this type of discounts need a commitment from you? And that's the reason why these are called committed use discounts. Using committed use discounts is recommended for workloads with predictable resource needs. You can say, I would need a virtual machine of this type for one year or three years. And, depending on the machine type and the GPUs you are asking for, you can get up to 70% discount. The discount that you would get

with committed use discount is higher than the discount that you would get with sustained use discounts. So if you're sure that you'd need a specific virtual machine for one year or three years, then it makes sense to go ahead and register for a committed use discount, because this gives you more discount than depending on a sustained use discount. Again, the committed use discount is applicable for instances which are created by Google Kubernetes Engine and Compute Engine and the same restriction that we talked about when we talked about sustained use discounts applies for committed use discounts as well. This does not apply for VMs created by App Engine flexible and Dataflow. Now how can you ask for a committed use discount? Let's go back to our virtual machines and in the menu for virtual machines the last option you'd see is committed use discounts. This is where you can actually go in and ask for a committed use discount. This is where you can actually go in and purchase a commitment. So I can go ahead and say, "purchase a commitment". And I can say, "I would want to purchase a specific commitment". As you can see in here, you can actually purchase a hardware or a software commitment. What we are interested in is a virtual machine commitment which is a hardware commitment. So you can give it a name, you can choose which region you'd want to have the commitment in, and you can choose the specific family you'd want to commit on, and then you can specify how much duration you'd want to commit on. You can also specify a little bit about the configuration of the machine you would want. Do you want codes, do you want memory? Do you want GPUs? Do you want local SSDs? And then you can go ahead and click purchase. I'm not going to click purchase in here because I don't really want to commit on it. What I would do is I would go ahead and say, "cancel". Once you register a commitment if you create any new instances of that type the discount is automatically applied. In a step we talk about committed use discounts. These are recommended for workloads with predictable resource needs. You know ahead of time that I would want to run a web application for a long time. In those kind of situations, you can get the best discounts by going for a committed use discount. I'll see you in the next step.
Welcome back. In this step, let's talk about preemptible virtual machines. What are preemptible virtual machines? Preemptible virtual machines are short-lived, cheaper compute instances. If you're aware of AWS, then preemptible VMs are very, very similar to sport instances. The preemptible instances can be stopped by GCP any time within 24 hours. The maximum time that you can run a preemptible instance typically is 24 hours. And you need to remember that GCP can terminate this instance any time within the 24 hours period as well. Just before GCP terminates your instance, you will get a 30-second warning. You can use this time period to save anything you might want to save. Now, when do you go for a preemptible VMs? You'd use a preemptible VM if your applications are fault-tolerant. You can stop and start them at any point in time, and you are very, very cost-sensitive. You'd want to run the workload with minimum cost and your workload is not immediate. If your users are accessing a specific application, then your workload is immediate. You'd want to run that immediately. However, if you have batch programs which you can run, let's say, within the next 15 days or within the next month, those are good workloads where you can go for preemptible VMs. So a good example is non-immediate, batch processing jobs. Now let's look at a few restrictions. Preemptible VMs might not always be available. Number two, there is no SLA that is guaranteed by Google Cloud platform. And your preemptible VMs cannot be migrated to a regular VM. You cannot convert a preemptible VM into a regular VM. And number three is you cannot automatically restart them. If they are stopped due to any reason, you cannot actually restart them again. And the last but very important one as far as this course is concerned is free tier credits are not applicable for launching preemptible VMs. So I would recommend you to not launch preemptible VMs at all. Now let's try and see how you can actually launch a preemptible VM. We're not going to launch it, we're just going to see where you can actually launch it. So I'm going to VM instances and you can go and say, "create" and this is the typical configuration we do. Now, where can I do preemptible instance? The place where you can configure that is in management security risks networking and salt tenancy. If you go further down, if you go to the availability policy, that is where you can actually configure the preemptibility. A preemptible VM costs much less but last only 24 hours. It can be terminated sooner due to system demands. And this is where you can actually set it to on. If you set preemptibility to on and launch an instance, then it becomes a preemptible instance. In this step, we looked at preemptible VMs. These are short-lived, very, very cheap compute instances which you can use to run non-immediate, fault-tolerant workloads. If you have a non-immediate batch program that you'd want to run at a very, very low cost, then you can opt for a preemptible VM. I'm sure you're having a wonderful time and I'll see you in the next step.
-: In this app, let's get a quick update about Spot VMs. What are Spot VMs? Spot VMs are nothing but the latest version of preemptible VMs. In earlier step, we talked about preemptible VMs. Spot VMs are the latest version of these preemptible VMs. Now, how is Spot VM different from preemptible VMs? The key difference is that Spot VMs do not have a maximum runtime. Earlier when we talked about preemptible VMs, we talked about the fact that preemptible VMs have a maximum runtime of 24 hours. You cannot run traditional preemptible VMs for more than 24 hours. However, there is no such restriction on Spot VMs. It does not have a minimum or a maximum runtime. Other than this, all the other features of Spot VMs are similar to traditional preemptible VMs. They can be reclaimed by Google Cloud at any point in time with a 30 second notice. When our Google Cloud is running short of resources,

they can send a termination notice and you have 30 seconds to clean up. Spot VMs, similar to preemptible VMs, might not be always available and these are also used at dynamic pricing. You can get about 60 to 91% discount compared to on-demand VMs. You cannot use free tier credits to create Spot VMs. I just wanted to ensure that you don't get confused by seeing Spot VMs in terminology somewhere. Spot VMs are nothing but latest version of preemptible VMs. The key difference between the older version of preemptible VMs and Spot VMs is the fact that there is no maximum runtime for Spot VMs. I'm sure you're having a wonderful time and I'll see you the next step.
Welcome back. In the last few steps, we were talking about a few discounts that are applicable for your Google Compute Engine. In this step, let's look at a few more important things around billing of your Compute Engine. It is very important to remember that you are billed for Compute Engine by this second. However, this is only after a minimum of one minute. So if you start a new VM instance you will pay for a minimum of one minute and thereafter you'd be billed by this second. Whenever you stop a Compute Engine instance you are not billed for compute. However, remember that if you have any storage that is attached with the compute instance, you'd still be billed for it. The first recommendation when it comes to reducing your billing is to always create budget alerts. How can you create budget alerts? If you go over to the navigation menu on the left hand side you should see something called billing. And in billing, you should see budgets and alerts. If you click that, you'd be landing on this specific screen where you can actually create a budget. Avoid surprises on your bill by creating budgets to monitor all your Google Cloud charges at one place. After you set a budget, you can create budget alerts to email billing admins and users when charges exceed a certain amount. So go ahead and create a budget. Let's create a budget right now. Create a budget. And I would call this budget as my first budget. And I'll choose the default, which are suggested for me. So I'll choose all projects, also visas, and I would say next. And over here I can specify a target amount. I'll say the target amount for me is let's keep it as low as possible. So let's just say I would say target budget is 10 rupees. And I would go ahead and click next. And if you click next, you would see alert threshold rules. So you can configure alerts at different thresholds. You can see that the defaults which are suggested are get an alert when 50% of the budget is utilized, which is five rupees. Get an alert when 90% of the budget is utilized and also when 100% of the budget is utilized. You can further customize this thresholds if you'd want. And if you scroll down you can configure the notifications as well. So make sure that you check this up. So email alerts to billing admins and users. This would ensure that you get an email when the cost exceeds your limits. And at the end of that, you can go ahead and click finish. Now, the other thing that you can do in addition to creating a budget alert, is a billing export. So you can actually take the billing and export it to a number of services. A little later we'll be talking about BigQuery, which is one of the big data services, which is offered by Google Cloud. Using billing export, you can actually send your billing data to BigQuery dataset, and using BigQuery you can run your queries as well. We'll talk a little bit more about billing export when we talk about billing in depth a little later in the course. So the best practice, as far as managing your cost is concerned, is to create budget alerts and also to make use of budget exports so that you are on top of billing all the time. We already looked at a few ways that you can save money with virtual machines. Choose the right machine type and the right image for your workload and also be aware of the discounts available. So be aware of the sustained use discounts, and if you want, you can actually register for a committed used discount and also understand that you can get discounts by using preemptable Virtual Machine Instances. In this step, we talked a little bit more about the costs involved with your virtual machines created using your Google Compute Engine. I'll see you in the next step.
Instructor: Welcome back. In last few steps we talked about availability scalability, resilience, and a lot of other factors. Another important responsibility of an architect is to keep the cost low. How can you keep the cost of your solutions as low as possible? Let's look at some of the things that you can do with Compute Engine and load balancing. The first one is auto scaling. Have the optimal number and type of virtual machine instances running. So choose the right family for your virtual machine based on the workload, and ensure that auto scaling is enabled. When auto scaling is enabled, you can increase the number of instances or decrease the number of instances based on the load. That would ensure that you are being really efficient. The other important thing is to understand sustained use discounts. Know that if you are running virtual machines for a long duration in a specific month, you would get sustained use discounts automatically. If you have predictable long term workloads. If you have workloads you know will be running for a year, or two years, or three years, you can go ahead and make use of committed used discounts. You can say, "I would want to commit on this for one year or three year period." And you would get discount for that. If you have noncritical fault tolerant workloads. If you have a workload you can run let's say, within the next month, and the workload would take only three hours to run. In those kinds of situations you can go for preemptible VMs as well. Preemptible VMs are cheapest and they're awesome if your workloads are non-critical and fault tolerant. So as an architect, you're always worried about costs. And these are some of the options that you have to reduce costs when you are building solutions around your virtual machines. I'm sure you're having a wonderful time, and I'll see you in the next step.

06 - Getting Started with Gcloud

Welcome back. Starting this step, let's look at the command line interface, which is offered by Google Cloud, which is called gcloud. gcloud is the command line interface to interact with Google Cloud resources. Most GCP services can be managed from the CLI, from the command line interface, using gcloud. You can talk to Compute Engine virtual machines, you can create managed instance groups, you can create databases, and a lot of other Google Cloud resources using gcloud. You can create, delete, update, read existing resources and also perform actions, like deployments. One important thing to remember is that there are other CLI tools also, which are specific to other GCP services. For example, Cloud Storage, which is the object storage service in Google Cloud, has a command line utility called gsutil. Cloud BigQuery has a command line util, bq. Cloud BigTable has a command line util called cbt. And when we are playing with ports and deployments in Kubernetes, we would be using kubectl. gcloud will be used to manage the Kubernetes clusters, but kubectl is used to manage the ports and the deployments. Now you don't really need to worry about the other CLI tools right now. The important thing that you need to remember is for almost 75% of the Google Cloud resources, the way you would interact with it from the command line is by using gcloud. How do you install gcloud? gcloud is part of something called Google Cloud SDK, and Google Cloud SDK requires Python to be installed on your local machine. All the instructions to install Cloud SDK and gcloud are present at this specific URL. If you do a right-click, Copy Link Address, and go to that specific page, you'll be able to see all the instructions, based on your specific operating system. The most important thing that you need to remember is you don't really need to install Google Cloud SDK on your local machine to be able to access gcloud. There is another approach that we can use which is called Cloud Shell. You can also use gcloud from Cloud Shell. How do we do that? You can go into your Google Cloud console, so you can go to cloud.google.com, click Console, and you'll get to one of these pages. And over here, you can click Activate Cloud Shell. So as it says in here, Cloud Shell, "Manage your infrastructure and develop your applications from any browser with Cloud Shell." Until now, we have been using Cloud Console. This UI, which we are making use of, is Cloud Console. The command line option that Google Cloud provides for us is called the Cloud Shell. And as it says in here, Cloud Shell comes pre-built with Google SDK gcloud. It also has a online code editor, and there are several utilities which are pre-installed, and it is free to use for all users. So let's go ahead and say Continue. To enable us to use Cloud Shell, Google Cloud Shell machine will be provisioned for us. This would take about a minute or so, and at the end of it, you'd see that we are ready with Cloud Shell. The preferred way for me to use Cloud Shell is in its own window, so what I would say is, Open a New Window. This is the icon you can click, and this would open Cloud Shell in its own window. Cool, I have my entire window where I can use Cloud Shell right now. You can configure the preferences you'd want, you can configure a theme, and also you can configure a text size. And I can go back to the original window, and close the Cloud Shell terminal session here. And in the complete window, I can go in there and type in, gcloud --version. And it would show the version of the gcloud that is being used. It'll also show all the tools that are already installed as part of gcloud. You can see that things like bq, cbt and gsutil are already installed in here. In summary, gcloud is the command line interface using which you can interact with Google Cloud resources. It is part of Google Cloud SDK. gcloud is pre-installed on Cloud Shell, and that's what we made use of, until now. Now what I would do is do a clear, and what I would want to run is gcloud init. This is the command to initialize, or re-initialize, Google Cloud. So let's go ahead and say gcloud init, and what is this? So it's asking me to authorize. I'll go ahead and say Authorize. When you use a gcloud init, and there is an existing configuration which is present already, the existing configuration is shown to you, and you'd need to choose whether you'd want to create a new configuration, or continue using the existing configuration. You can see the different settings from the current configuration in here. You can see that the current project which is assigned, is glowing furnace 304608. What is this? If I go back to Google Cloud platform and do a Refresh, and go to myFirstProject, you can see that this is the ID of myFirstProject, glowing furnace 304608. Each project is assigned a globally unique ID, and that is the ID which is configured in here as part of my default configuration that was provided by Cloud Shell. What we'll do is we'll act as if there is no default configuration, we would want to create a new configuration. So I would say, Create a New Configuration, so two. And now it asks, Enter Configuration Name. So I would want to have a specific configuration that I would want to use, and I'll call this My Default Configuration, and press Enter. Next, it would ask me, what is the account that I would want to use with this specific configuration? I can say I would want to use the existing account itself. If you are running from your local machine, then you would want to enter two, and you can login with your new account. But over here I'm already authenticated with the earlier account, so I'll go ahead and use one. And next you can see that it is asking me, what is the project you would want to use? So first, account, next, project, I'll go ahead and say, I want to use the first project, glowing furnace 304608. You can see that it says, "Your current project has been set to glowing furnace 304608." Do you want to configure a default Compute region and zone? I would say I would want to configure a default Compute region and zone, as well. Important thing to remember is this is the default Compute region and zone when I'm working from this specific configuration in Cloud Shell. This is not going to change the default Compute region in Cloud, which is configured on Google Cloud. This is only a local setting, so I would say

Yes, and this would configure a local setting for region and zone. You can see that it brings in a list of all the regions. I'll just choose one randomly, I'll say one. One thing to remember is the list of things which is shown in here is not actually regions, but this is these zones. This is a list of zones, from which I chose one of them. I entered one, and it shows default zone. So what we have done until now is we use gcloud init to initialize or re-initialize gcloud. This will authorize gcloud to use your account credentials. This also allows us to set up configuration. We have set up our current project. We have set up the default zone right now. Now what we want to do is we want to list our configuration. What is the current configuration that we are making use of? The command we can make use of is gcloud config list. So let's go ahead in here, and type in gcloud config list. So gcloud, space, config, space, list. And now you can see the current configuration. You can see that the default region that I have configured in here is US East-1, and this is applicable only for Compute. And you can see that the zone, the default zone which is configured in here, is US East-1-B. And you can see the account details, and you can see the project. So the default account is this, and the default project is this. As far as the exam is concerned, there are four important things that you need to remember. Compute.region, compute.zone, and in the Core, you would see that account and project. So these are important configurations that you need to remember. In this step, we talked about the fact that gcloud is part of Google Cloud SDK, and the fact that gcloud is pre-installed on Cloud Shell. We saw gcloud init, which is the command to initialize or re-initialize gcloud with our account, and also configure the current project and default zone. We also saw the command gcloud config list, which lists all the properties of the active configuration. I'm sure you're having a wonderful time, and I'll see you in the next step.
Instructor: Welcome back. In this specific step let's look at G Cloud Command Structure. When you're playing with services whenever you're playing with compute or databases or storage, this is the typical command structure which which you'd be making use of with G Cloud. The command structure is G Cloud. What is the group? What is a subgroup and what is the action? Now, what is group and subgroup? A group might be something like config or compute or container or data flow or functions or IAM. Until now we have been playing with compute instances. So for example, compute is the group. Which service group are you playing with? Subgroup is the components of that specific service. If I'm playing with compute, inside compute you can either be playing with instances or images or you might be playing with in instant templates or you might be playing with machine types or you might want to list the region which are under compute or the zones which are under compute. Which subgroup of the service do you want to play with? That's the subgroup. And you can take actions, create, list, start or stop or describe or so on and so forth. What do you want to do? Let's see a few examples. G Cloud compute instances list. Let's try and run them and do a clear G Cloud compute is the group. Instances is the subgroup and list is the command that I would want to execute. G Cloud compute instances list. Right now, I don't really have any instances running. And what would it do? It would say, "Hey Ranga, you don't really have any instances running." It's taking a little while. Cool. At the end it comes back and says, "Listed zero items." Right? So that's cool. Now, if I want to create a new instance, I can say G-Cloud compute instances, create, right? Is this sufficient? Let's go and try enter. It says, "Nope, it's not sufficient." The error it says is, "Hey you're trying to create an instance but there is no name that you are attaching with it.' An instance name is mandatory. Okay, that's cool. So I'll say G Cloud compute instance, create my first instance from G Cloud. Let's press enter right now. What would happen? G Cloud compute instances create my first instance from G Cloud. Let's see what would happen. Cool. It's creating a new compute instance for us. You can see that it is making use of the region US East one B, where is it coming from? Let's do a G cloud config list. You can see that the default zone that we have configured right now in our configuration is US East one B and that's the reason why it's picking up that zone and using that to create this specific instance. The other interesting thing is the default machine type in here right now the default machine type is N one, standard one and you can see also the internal IP and the external IP and the status. Let's do a clear and let's do a G Cloud compute instances list now. So G Cloud compute instances list, what does it do? It would list the existing compute instances. We have just created one. So this would list the instances that we have created. Fingers crossed it's taking a little while. It's taking almost 30 seconds to list this down. And over here you can see a lot of details, right? So you can see the name zone, machine type, internal IP external IP and things like that. So if you'd want to see the internal IP and external IP which is assigned to a specific instance, all that you need to do is to say G Cloud compute instances list. Now you can also go ahead and do G Cloud compute instances describe. So the describe command is to describe a specific instance. So I can go ahead and say my first instance from G Cloud I can go and do this. So this would actually, Oops, I have a typo. It should not be G code compute. It should be actually G Cloud compute. Yep, I hope that's right. Yep, you can see that it describes all the details around that specific instance. It gives me a lot of details around it. So if you go further up, you'd see that there is a lot of information about that specific instance which is printed. Let's not worry about all those details. The important thing for you to remember is if you wanna get all the details about a specific instance you can use the described command just like we created the instance. You can also delete the instance by using the delete command. So G Cloud, compute instances, delete, and you can pass in the name, my first instance from Google Cloud. What would happen? It's asking me, "Do you want to really delete this

instance?" I would say yes, and go ahead and delete this. So this would go ahead and delete the instance which we have created. You can see that even from the command line, it's very very easy to play with our compute engine. We can easily create instances and delete them. Now let's look at a few more other commands that you can run. So G Cloud, compute zones list, regions list, machine types list. So you can actually go ahead and try this. So G Cloud compute zones list, what would it print? It would print the different availabilities ones which are present in here. So what are the different compute availabilities zones which are present? You can also do G count compute regions list. It would print what are the different regions that are present as part of the compute service. So you can see these are the different regions which are present as part of the compute service. Similar to that, you can also print machine types. So what are the different machine types which are present inside compute? This would bring up a huge list of things. So G-cloud compute machine types list would usually be a huge list. You can see that it's printing the machine type and which region it's available in. As we discussed earlier machine types can vary based on the region and the zone and that's the reason why it's printing machine type, region and a few more details about that specific type. And to be able to clearly understand that, what I would want to do is I would want to filter down So let's say I only want to print the machine types which are present in a specific region. How can I do that? I can do a hyphen, hyphen filter and what do I want to filter by? I want to filter by the zone and which zone do I want to filter it by? I want to say Asia and Southeast. You can pick the region which you see last in here. So I'm picking up Asia hyphen Southeast two, hyphen B. Let's filter only those machine types which are in this specific zone. So zone column, the value of the zone. Let's hope this would bring us a little less number of machine types, right? These are all the machine types which are available in this specific zone. You can see that the numbers which are in here are the CPUs and the memory. How many virtual CPUs are attached with that specific machine type, and what is the amount of memory which is present along with it. Now let's say you'd want to filter from multiple zones. How can I do that? The way I can do that is by putting the values in parenthesis. So inside parenthesis I put one value. Right now I just have one value within the parenthesis. In addition to this, I would want to add in another zone. So I can say Southeast two hyphen C, so B and C, Asia hyphen southeast, two hyphen B Asia hyphen southeast, two hyphen C. So I'm going to list machine types from two zones. Oops, there is an error. The error is that we need to put this value in double codes so half and half and filter within double codes and another double code in here and press Enter, fingers cross. I hope this would bring me all the values which are present all the machine types which are present in Southeast two B and southeast two C. Cool, that works. So this brings me all the instant types which are present in two B and two C. You can see that in here. That's cool. In this step, we understood the High Level Command Structure which is used by G Cloud and we started playing around with compute service. I'm sure you're having a wonderful time and I'll see you on the next step.

Welcome back. Earlier in the course we played with Cloud Shell. There are a few important things that you'll need to remember about Cloud Shell. Whenever you create a Cloud Shell it is actually backed up by a virtual machine instance. There is a Google compute engine virtual machine instance which is automatically provisioned whenever you launch Google Cloud Shell. This virtual machine is attached with 5 GB of free persistent disc storage and this is what is provided as your home directory. So inside your home directory you can start up to 5 GB of files. This is prepackaged with latest version of Cloud SDK, Docker, and a number of other software. Any files that you store in your home directory persist between sessions so if you have any scripts or if you have any user configuration files as long as they're stored in your home directory they persist between different sessions. If you don't use Cloud Shell for more than 20 minutes if you're inactive for more than 20 minutes the VM instance is automatically terminated. If you have made any modifications outside the home directory, they'll be lost. A very important thing to remember is that if you don't use Cloud Shell for 120 days then even content in your home directory will be deleted. Another important use case for Google Cloud is to SSH into virtual machines using their private IP addresses. So if you have a virtual machine that you'd want to SSH into and perform certain actions you can use the private IP addresses of the virtual machines and use Cloud Shell to SSH into them. These are some of the important things that you need to remember about Cloud Shell. I'm sure having interesting time and I'll you in the next step.

07 - Getting Started with Google Cloud Platform Managed Services

Welcome back. In the previous steps we manually configure the virtual servers, we manually set upload balancing, and we had to do a lot of things manually to get our applications up and running. Is that the only way you can run applications in the cloud? The answer to that is no. All the cloud providers provide a number of managed services. In this section, let's get a 10,000 feet overview of managed services. Let's get started. We want to continue running applications in the cloud the same way you run them in your data center. Or are there other approaches? To understand this better you need to understand some of the terminology which is typically used with cloud services. Infrastructure as a service, platform as a service, function as a service, container as a service, serverless, and there are tons of other terminology as well. Do not worry if this terminology is new for you. In this section, let's get on a quick journey to understand all these terminology. I'll see you in the next step.

Welcome back. Let's get started with Infrastructure as a Service, IaaS. When you're using Infrastructure as a Service, you are only using Infrastructure from the cloud provider. A good example of this is using a virtual machine to deploy your applications or databases. If you're using AWS, you might be using EC2. If you're using Azure or Google Cloud, you might be creating virtual machines and then you would manually deploy your applications, or you would manually set up your database on that virtual machine or EC2 instance. When you're using Infrastructure as a Service, you are responsible for application code and application runtime. You are responsible for configuring Load Balancing between your virtual machines. You are responsible for Auto Scaling your application. You are responsible for the operating system, and making sure that your operating system is upgraded and properly patched. You're responsible for availability of your application. Take the case of deploying a Python application or a Java application or a Node application. When you're doing Infrastructure as a Service, you are responsible for taking the virtual machine, choosing the right operating system for it, installing the required application runtime. For example, you'd want Java or Python or Node.js, you would need to install that. And then you would need to install your application code, and set up the configuration for it. So you are responsible for everything about the virtualization layer. The cloud provider is responsible for providing the physical hardware, networking, and making sure that you have access to some virtualization software so that you can create your virtual machine instances. For all the remaining things, you are responsible. This is how we deployed applications in our data centers until a few years back. The alternative to this is to go for Platform as a Service, user platform which is provided by cloud. When you're using Platform as a Service, cloud provider is responsible for the operating system including making sure that the operating system is upgraded and patched. The cloud provider is responsible for the application runtime. If you want to install Java or Python, the cloud provider can take care of that for you. The cloud provider is also responsible for Auto Scaling, availability, and Load Balancing. You are only responsible for the configuration of the application, the configuration of the service and the application code, if you have application code. Good examples for PaaS or Platform as a Service include Elastic Beanstalk from AWS, App Service, which is provided by Azure, and Google App Engine. When you're using any of these services, all that you need to focus on is the application code. If you're deploying a Java application to any of these, you just provide the code to the specific service. The cloud service would ensure that the application runtime is properly configured the OS is properly patched, and you get all the features like Auto Scaling, availability, and Load Balancing for free. Now, let's look at a few varieties of Platform as a Service. There's something called Container as a Service where you have containers running on your platform instead of the applications. You also have FaaS or Function as a Service where you have functions running instead of apps. And also there are a variety of other platform as services. Platform as a Service is not limited to compute services alone, there are Platform as a Service offerings for databases, both relational databases and NoSQL databases. For examples of relational databases in the cloud are Amazon RDS, Google Cloud SQL, Azure SQL databases, and there are variety of other things also available as Platform as a Service. For example Queues, Amazon SQS, Google Cloud Pub/Sub, Azure Queue storage. The cloud providers also provide you with a number of platform as service offerings for artificial intelligence, machine learning, and also performing operations. In the start, we got started with Infrastructure as a Service and Platform as a Service. When you're using Infrastructure as a Service, the cloud provider only provides the physical hardware, networking, and the virtualization layers. You are responsible for everything above that, making sure that the right OS is installed, making sure that the application runtime is installed, making sure that your application is properly deployed. And you are responsible for taking care of Auto Scaling, availability, and Load Balancing. And when we are going for the cloud, typically we would offer Platform as a Service offerings. When you're going for a Platform as a Service offering, the cloud provider is responsible for everything up to the application runtime. You are only responsible for configuration of the application, configuration of the managed service. And if there is code involved, then you are responsible for the application code as well. And we talked about the fact that there are a variety of Platform as a Service offerings that cloud providers provide. We referred to Container as a Service, Function as a Service, databases, Queues, operations and a lot of other things. In next step, let's look a little deeper into Container as a Service. Why do we need containers, and what are the typical features that Container as a Service services provide? I'll see you in the next step.

Instructor: Welcome back, in the step. Let's look at containers. Why do we need containers? What is container as a service? And to understand that, we will get started with microservices. Enterprises are heading towards microservices architectures. Instead of building a large monolith application the focus is to build small microservices. The biggest advantage with microservices is the flexibility to innovate. You can build applications in different programming languages. For example, you can build the movie service in Go. You can build the customer service in Java. You can build the review service in Python. You don't really want applications built with a variety of languages in the same enterprise but you have the flexibility to do that. But a byproduct of going towards the microservices architecture is that your deployments become complex. The way you deploy your Go application might be different from the way you deploy your Java application. How do you ensure that we have one way of deploying microservices which are built in different languages? That's where we talk about containers. One of the popular container related tools is Docker. What you can do when you're using containers is you can create docker images for each of your microservices. The docker image contains everything that you need to run a microservice. It contains the application run time. Let's say you're running a Java application you need a JRE or JDK. Let's say you're running a Python application or a Node.js application. You need the corresponding run time. The docker image also contains application code and the dependencies that the application needs. So the docker image contains everything that you need to be able to run a microservice. You can create docker images for Python applications Java applications, Node.js applications and once you have a docker image you can run it the same way on any infrastructure whether you're running it on your local machine or your corporate data center or in the cloud. Once you have a docker image, you can run it anywhere and create containers from it. A friend of mine says, when I'm using Docker I don't really have problems like it runs on my local machine. Because you have using docker images you can use the same docker image on local machine and your deployment and environment, therefore you'll not have problems like it works on my local. Now, what are the advantages of going with containers? Docker containers, in general containers are lightweight. They are lightweight because compared to virtual machines they do not have a guest OS. Before the emergence of docker if you'd want to do any virtualization, you used to use virtual machines and in virtual machines, in addition to the host OS there was another guest OS which was present. However, when you're using docker, you don't really need a guest OS you have the cloud infrastructure or any kind of infrastructure, and on top of it you have a host operating system installed. On top of it is where you would install your docker engine or container run time. Once you have that, you can run Java containers, python containers, or Node.js containers on top of it. Because there is no guest OS which is needed docker containers are generally considered to be light to eight. Another advantage is that the containers are all isolated from one another. If there's a problem with container one it'll not impact container two and container three. These are isolated from each other and the most important advantage is cloud neutral. Once you have a container image you can run it on your local machine, your data center and in any of the cloud providers you can run it in AWS, Azure and Google Cloud, for example. Each of the cloud providers provide a variety of options to run your containers. Now you have built container images for your microservices and you're able to easily run containers with them. You have built a container for microservice A, B, C, and D. And now you'd want to actually manage the deployment of these containers. You'd want to ensure that you'd be able to auto scale the number of container instances that are running based on the load. You might have requirements saying, I want 10 instances of microservice A container, 15 instances of microservice B container, and so on and so forth. Just having containers is not sufficient. You would need some kind of an orchestration around these container, and that's where there are a number of container orchestrator solutions. If you have heard about Kubernetes, then Kubernetes is one of the most popular open source container orchestrator solutions. When you're using these container orchestrator solutions you can say, this is the container image for microservice A and I would want 10 instances of it. You can say, this is microservice B, I'd want five instances of it or 15 instances of it, and the container orchestrator tool would manage the deployment of these containers into clusters. Each of these clusters can have multiple servers and these container orchestrators typically offer a number of features. Auto scaling, you can say, this is the contain image for microservice A, and I expect a lot of load on it so I would want to auto scale, so based on the number of requests which are coming into microservice A the container orchestrator can scale the number of instances of that specific container. Service discovery is very, very important feature for microservices. You might have 10, 15, 20, or hundred microservices. You don't want to hardcode the URLs of each microservice in another microservice. That's where the concept of service discovery comes into picture. Each microservice can ask the container orchestrator for the location of other microservices and thereby you don't really need to hardcode the URLs. As soon as I start talking about multiple containers. You need to also talk about load balancing. Once I have multiple containers I want to distribute the load between them. Container orchestrators also provide load balancing. You also want resiliency. If one of the instances of a microservice is not working properly you'd want the container orchestrated to identify that and replace it with a working instance. That's where you can configure health checks and the container orchestrator can execute frequent health checks and replace failing instances. This is also called self feeling. Not only that, you also want zero downtime deployments. You might want to go from version one to

version two of microservice A, however you don't want any downtime. Container orchestrators also provide a number of strategies to release your new versions of software without downtime. Kubernetes is one of the most popular container orchestrator tools. All the cloud providers provide Kubernetes managed services so EKS, which is provided by AWS or Elastic Kubernetes service. AKS, Azure Kubernetes Service. You also have GKE Google Kubernetes Engine which is provided by GCP. In addition to the services around Kubernetes Cloud providers also provide other services around containers. AWS provides ECS, Elastic Container Service. Google also provides Cloud run which is a simple service to run simple containers. In this quick step, we got a high level overview of containers, container orchestration and some of the important container as a service offerings in different cloud providers. We'll talk about this little more as we go further in the course. I'm sure you're having an interesting time and I'll see you the next step.

Welcome back. In this step, let's get started with serverless. You have heard this term a number of times. So, what is serverless? When do you go for serverless? What do you think about when we develop an application? In addition to choosing the language or the framework that you would want to develop the application with, you would also start thinking about where to deploy the application. What, what kind of operating system should I deploy my application into? How should I take care of scaling availability and all the nonfunctional features around my application? What if you don't really need to worry about server, server configuration, scaling availability and you can just focus on your code and your application? That's serverless. The important thing to remember is serverless does not mean no servers. Even when you are using serverless, you are using servers in the background. The only difference is that the servers are not visible to you. You don't have any visibility into the servers on which your code is running. Now, what is serverless for me? The important characteristics of serverless for me are, number one you don't worry about infrastructure. So, you have zero visibility into the infrastructure where your application is running. You don't know which server is being used to run your application, and you get flexible scaling and automated high availability for free. And another important characteristic of serverless for me is pay for use. Ideally, if there are no requests for your service, you should pay zero cost. So, serverless is all about allowing you to focus on code and the cloud managed service taking care of everything that is needed to scale you a code to serve millions of requests. And in addition, when you're using serverless, you pay for requests and you don't pay for servers. So, you pay for number of invocations of your function or number of invocations of your application and you don't pay for how many servers are used to deploy your application. Now, what are the examples of serverless services? Good examples are all the function as a service services AWS Lambda in AWS, Azure functions in Azure, and Google functions in Google Cloud platform. In this step, we got a 10,000 feet overview of serverless. We'll talk more about serverless as we go further in the course. I'm sure you're having an interesting time, and I'll see you the next step.

Welcome back. In this step, let's get a 10,000 feet overview of the different managed services for compute that are offered by GCP, the Google Cloud Platform. We already talked about Compute Engine earlier. It is used to create Virtual Machines that can scale globally. And where does it fit in? It fits in the Infrastructure As A Service category. If you're making use of a Compute Engine, you are responsible for choosing the operating system. You are responsible for installing the application run time, and you are responsible for installing your application or installing your database and things like that. So Compute Engine is Infrastructure As A Service. The next managed service is Google Kubernetes Engine. Earlier we talked about container orchestration, and Kubernetes is the most popular container orchestration tool and Google Kubernetes Engine is Google's Kubernetes service. One important thing about Kubernetes is that when you're using Kubernetes, you need a cluster. So you need advanced cluster configuration and monitoring. You create a cluster. The cluster contains a number of nodes or number of instances and you deploy your microservices using Kubernetes into the cluster. So Google Kubernetes service is a Container As A Service offering from Google Cloud. The next important compute service is App Engine. If you have a Java application or a Python application or a Node application if you want to easily deploy it to GCP without even worrying about creating a container, you can go with App Engine. You can build highly scalable applications on a fully managed platform using open and familiar languages and tools. App Engine fits into the category of Platform As A Service. However, if you have a container you can also deploy simple containers to App Engine. It does not provide complex container orchestration features like Kubernetes, but it does provide a few simple features to run your containers. So App Engine provides a few Container As A Service characteristics, and also you'd see App Engine being referred to as Serverless Platform by Google because if you have no load, then App Engine Standard can go down to zero instances. For example, if you have a Java application with Adhoc load, you get a few requests. It is idle for a long time, then you get a few requests. In those kind of situations, you can go for App Engine Standard because when you don't have request, App Engine Standard can go down to zero instances. The next important managed service is Cloud Functions. This is an example of Function As A Service. And this is a true serverless offering from Google Cloud Platform. You can build simple event driven applications you can build simple event driven applications using the functions which you create using Cloud Functions. If you want to do something immediately, as soon as there is a message on the queue, you can create a cloud function to listen on the

queue and react to it. If you want to do something as soon as an object is uploaded into cloud storage, you can do that using a Cloud Function. The next service is a relatively new offering from Google Cloud Platform. This is Cloud Run. Cloud Run can be used to develop and deploy highly scalable containerized applications. So Cloud Run is also a Container As A Service offering. However, the difference between Kubernetes and Cloud Run is that Cloud Run does not need a cluster. Kubernetes is recommended for complex microservices architectures. However, Cloud Run is recommended for much more simpler architectures. So if you have simple container applications that you'd want to run, you can try Cloud Run. In this step, we got a 10,000 feet overview around the different compute managed services that are offered by GCP. I'm sure you're having an interesting time and I'll see you in the next step.

08 - Getting Started with Google Cloud App Engine
Speaker: Welcome back. Welcome to this new section on App Engine. App Engine is one of the most important managed services provided by Google Cloud Platform. Actually, App Engine is one of the earliest managed services in the cloud. App Engine was first released by Google Cloud Platform as far back as 2008. Yes, I'm talking about 2008 which was when the financial crisis started. So it's almost a decade and half back that the first version of App Engine came about. So let's get started with App Engine in this section. App Engine is the simplest way to deploy and scale your applications in GCP. It provides end to end application management. It provides you with a lot of features. It supports Go, Java.net, Node.js, PHP, Ruby and Python using preconfigured run times. You can also use a custom runtime and write code in any language. App Engine also supports running containers so you can create a container, use a custom run time and write code in any specific language. App Engine has really really good connectivity with other Google Cloud services. If you want to talk to a database like Cloud Sequel or if you want to talk to object storage like cloud storage or if you want to talk to a queue, even that option is provided by App Engine. There are no usage charges for App Engine itself. However, you need to pay for the resources provision. If you have provision compute instances through App Engine then you'd pay for those. Some of the important features of App Engine include automatic load balancing and automatic scaling. So you can say, this is the application I would want to run and it can auto scale to as many users as you would want. App Engine would automatically increase the number of instances where your application is running and also automatically load balance between them. App Engine can take care of the platform updates and also monitor the health of the application. App Engine also provides versioning of your application. You can have multiple versions of your application running and you can manage traffic between them. So App Engine supports traffic splitting. You can have two versions, three versions or four versions of the application running and you can split the traffic between them. You can say, I would want to send 30% of my traffic to the latest version. I want 70% of the traffic to go to v3. So things like traffic splitting can be done very very easily using App Engine. Let's quickly compare. Compute Engine versus App Engine. So Compute Engine is infrastructure as a service. It provides you with a lot of flexibility. Along with the flexibility comes more responsibility. You're responsible for choosing the image. What OS do you want on the Compute Engine? What software do you want to install? I would want to install Java. I would want to install Python. I would want to install Node. You need to control that. You need to also decide what hardware you'd want to run. You also need to configure access and permission to like certificates, firewalls, everything you need to handle. And you are also responsible for the availability, scalability of your application. However, with App Engine, it's platform as a service. It's serverless. You have lesser responsibility. However, you have lower flexibility. So if you'd want to add a GPU, you cannot do that with an App Engine. So if you have a simple Java application, a Python application or a containerized application already present and you'd want to easily deploy that to the cloud, then App Engine might be the right option. However, if you'd want a lot of flexibility, you'd want a specific operating system, you'd want specific hardware with a GPU attached. And if you'd want fine grained permissions then Compute Engine might be the right option for you. In this step, we got a 10,000 feet overview of App Engine. To understand it further, let's get our hands dirty in the subsequent steps. Let's dig deeper into App Engine in the next steps. Welcome back. Let's start with understanding some of the important concepts around App Engine. The first concept that you need to understand is the concept of the different environments that App Engine provides. App Engine provides two different kinds of environments, Standard and Flexible. When you're using Standard, your application runs in language specific sandboxes. So there is a sandbox for Java, there is a specific sandbox for Python and your applications would run in that specific pre-configured sandbox. You can call this an advantage or disadvantage of going for a sandbox. You have complete isolation from OS disk and other apps. You don't need to worry about any of them. All that you need to worry about is your specific application code. The language specific sandbox is automatically provided by App Engine Standard. App Engine Standard comes in two versions, V1 which supports Java, Python, PHP, and Go. And these are the older versions of each of these software. So the older versions of Java, Python, PHP and Go are supported by V1. And there are a few specific restrictions for Python and PHP runtimes. You have restricted network access and there are specific white-listed extensions and libraries which are allowed. So only for Python and PHP runtimes, there are a few more restrictions. You cannot have any network access and there is a specific set of extensions and libraries that are available. You cannot use anything else when you're using Python or PHP runtimes. However, there are no such restrictions for Java or Go runtimes, even with V1. There is also a V2, so this is the newer version of Standard which is recommended by Google. So if you're creating a new application and if you're planning to use Standard, then you should use the V2 version of Standard. This would support all the newer versions, so Java, Python PHP, NodeJS, Ruby, Go. All these are supported by V2. This provides you with full network access and there are no restrictions on language extensions. So Standard is when you'd want to run applications in language specific sandboxes. The other option is to run containers. That's Flexible. That is also called App Engine. This is also called App Engine Flexible. So application instances run within Docker containers. One important thing to remember is that when you're using Flexible, you're making use of compute engine virtual

machines. Flexible supports any runtime. There are no restrictions as such. It has built in support for certain specific runtimes. So if you have Python, Java, NodeJS, Go, Ruby, PHP, and .net, the built-in support can even build the container for you before deploying it to App Engine Flexible. However, you can use any runtime as long as you can build a Docker image for it. And with Flexible, you also get access to background processes which are running on the virtual machine, and also you can attach local discs with your virtual machines. In this step, we talked about Standard versus Flexible. These are the two kinds of environments which are provided by App Engine. App Engine Standard runs applications in language specific sandboxes. There are a lot of restrictions when you are talking about Standard. App Engine Standard V2 supports Java, Python, PHP, NodeJS, Ruby, and Go. If you want to run a C++ application, you cannot do that in Standard. If you want to run a C application, you cannot do that in Standard You have to create a container and then you have to run it in App Engine Flexible. App Engine Flexible is where you would actually run your applications within Docker containers. You have a lot of flexibility when you are using App Engine Flexible. Because it supports any runtime, it has built-in support for a certain set of languages. However, if you're able to build a container image you'd be able to run it using App Engine Flexible. And because your App Engine Flexible applications are deployed to compute engine virtual machines, you get access to the background processes and local lists that are attached with your virtual machines. I'm sure you're having a wonderful time and I'll see you the next step.

Welcome back. In the previous step, we talked about the different and environments that are provided by App Engine Standard and flexible. And in this step let's talk about the application component hierarchy which is offered by App Engine. Whenever we talk about App Engine you need to remember about application surveys and versions. So what is application, what is service? What is version, application is kind of the container for everything. You can create only one application per one project, and the application is the container for everything that you would create as part of your app engine. For example, if I would want to deploy three microservices then the first thing is I have one application in a specific project, and underneath it I can create different services for each of the microservices. If you have a full stack application then you can also create services one for the backend and one for different end. So services are kind of, what are the microservices or what are the different app components that you would want to run as part of your app engine? You can have multiple services in a single application. Each service can have its own settings. Earlier services were also called modules. Underneath the service you can have multiple versions of each service. A version is nothing but a specific version of your service associated with code and some configuration. So I can have a V one of the service V two of the service V three of the service so I can have multiple versions of the same service. Each version can run in one or more instances. So you can see that for this specific version we have two instances running. And for this version, there are no instances running and you can have multiple versions existing at the same time. So you can have V1 and V2 of this application running at the same time and you can even split traffic between them. So you can say, I would want 10% of the traffic to go to V one and 10% of the, and 90% of the traffic to go to v2. So you have options to roll back and split traffic as part of your app engine. In this step, we focused on the high level structure of App Engine. When we are talking about App Engine you can only have one application per project and this application acts as the container for all the services that you'd want to create as part of your project. And each service can have multiple versions and you can easily split traffic and roll back to specific versions. Let's talk a little bit more about it when we get to the hands on part of this particular section. I'll see you in the next step.

Welcome back. In this step, let's quickly compare the different App Engine Environments. We talked about App Engine Standard and Flexible. What are the different features that are offered by each one of them? Let's look at them right now. Let's start with pricing. How do you price App Engine Standard? App Engine Standard is based on the instance hours. You don't really need to worry about CPU or memory or any of such configuration. However, when it comes to App Engine you are priced based on the amount of CPU memory that is used by your virtual machines. You are charged based on the persistent discs or the storage that you attached with your virtual machines. There are three different types of scaling that you can do with App Engine, Manual, Basic and Automatic. Manual is when you'd want to have full control so you'd want to manually update the number of instances. Basic is when you have ad hoc workloads so I have some load coming in and maybe for a few hours I don't have anything and then I have a little bit more workload coming in. And Automatic is recommended when you have continuous flow of traffic. So based on the number of users you'd want to automatically scale up and scale down. We'll talk a little bit more about scaling options a little later. However the important thing to remember is that App Engine Flexible does not support basic. It only supports manual scaling and automatic scaling. App Engine Standard supports all three. The other important thing is scaling to Zero. Can you scale down to zero instances with Standard? Yes, you can scale down to zero instances with Standard. If there is no load, you can go down to zero instances. However, with Flexible, there is no such option. You need to have a minimum of one instance running all the time. If you'd want to serve graphic you need to have at least one instance running all the time. The Instance Startup for Standard is seconds and Instance Startup time for Flexible is in minutes. So if you have applications which you'd want to

quickly start then Standard might be the way to go. And because the Instance Startup is in seconds, Standard supports rapidly scaling. You can rapidly scale up and rapidly scale down. However, with Flexible because it takes a few minutes to scale up, rapid scaling is not really supported. The maximum request timeout, for standard is between one to 10 minutes, for flexible, it can go up to 60 minutes. So if you have application requests which go on beyond 20, 30 minutes, then flexible might be the way to go. You can access local risk from standard except for Python and PHP run times. You can use this /temp folder. With Flexible you can attach discs. However important thing to remember is that the disc storage is ephemeral, it's not permanent storage. Once the instance is restarted you'd lose everything that is on the local storage. Another important feature that App Engine flexible allows is SSH for debugging because in the background there is a virtual machine you can SSH into it. The idea behind this step is to quickly compare the different environment provided by app engine, standard IT versus flexible. I'm sure you're having a wonderful time and I'll see you on the next step.
-: Back. In the last step, we talked about the different options for scaling instances. We talked about manual, automatic, and basic. We talked about the fact that app engine standard supports only three. However, app engine flexible supports only two of those. Let's discuss more about scaling instances in this specific step. So, automatic is to a automatically scale instances based on the load. This is recommended for continuously running workloads. If you have, if you have an application for which you'll continuously have users, automatic scaling is recommended. So, you can auto scale based on a variety of configurations. You can say this is a CPU utilization target. You can configure a threshold. You can also say, this is a throughput that I would want. You can configure a throughput threshold. You can configure maximum number of concurrent requests. How many maximum concurrent requests do you want to serve from a specific instance? So, you can auto scale based on any of these parameters. And you can also configure, what is the maximum number of instances that you'd want to run and what is the minimum instances that you'd want to run. So, that's automatic for continuously running workloads. The next option is basic. Basic is when you'd want to create instances as and when requests are received. These are recommended for ad hoc workloads. When there is no load, instances are completely shut down. So, instances are shut down if there are zero requests. So, if you have ad hoc workloads, and it's very important for you to keep your cost low, then basic scaling might be a good option for you to consider. However, the disadvantage of basic scaling is the fact that high latency is possible. Why is it's there high latency, because it's possible that there are no requests that are being served by a basic scaling instance right now and therefore a new instance has to be created and then your request would be served. Another important thing to remember for basic is the fact that it is not supported by app engine flexibility, it is only supported by app engine standard. When you're configuring basic scaling, you can configure what is the maximum number of instances you can go up to and what is the idle timeout. So, if you have no request for a specific amount of time period, then you'd go down to zero instances. The last option is the manual one. This is the easiest one as far as configuration is concerned. However, there is a lot of manual work involved. You can configure how many instances to run and you can keep adjusting it manually over a period of time. If you have a magic wand. And if you can exactly predict what is the load that would be on your application then you can go for manual scaling. In this step, we looked at the three options for scaling instances in app engine, automatic, basic, and manual. We talked about the fact that basic is not supported by flexible environments. You'd go for automatic, for continuously earning workloads. You'd go for basic if you have ad hoc workloads and you'd want to keep your costs as low as possible by reducing the number of instances to zero. And if you have the magic wand and you can predict the load, you can go for manual option. I'm sure you're interesting time.
Instructor: Welcome back. Are you bored with all the theory that you have been hearing about App Engine until now? Let's get started with playing with App Engine right now. Let's deploy an application to the cloud using App Engine standard. The first thing I would do as usual is to go to our Google Cloud platform. And what we do now is we would create a new project for this specific App Engine. The thing is, I can actually create an App Engine in this specific project as well. That should not be a problem. But I would want to show the creation of the project and all that process as well to you. So let's create a new project and create an App Engine in there that also allows us to delete the project and delete everything that is present in there. So your App Engine would be automatically deleted as soon as I delete the project. So instead of deleting individual resources which are present in the project, let's try and shut down a project and see that in action too. So let's go ahead and start creating a project. So I'll go into the project screen and I'll say new project. So, and I'll call this App Engine, or I'll call this Learning App Engine Project, Learning App Engine Project. So you can see that there is a limit on how many projects that we'll have. We can have up to 21 projects. So that's a lot. And you can see that we are under no organization, because we are using a free data account. So all that is fine. So let's go ahead and say create. So the project would be created in a little while. So over here, if I go in and select learning app project, you'd see that there is a new project over here. And over here I can switch back and forward between the projects. So my first project. Now, let's get back to focusing on App Engine. So inside this Learning App Engine project, I would want to use App Engine. What would I need to do first? I would need to go to App Engine, right? So when I go to App Engine, there'd be one thing that would happen. What would

happen? Think about it. Whenever you'd want to make use of any service within a new project, the first thing that you would need to do is to enable the APIs for that specific service. So inside this project, if I actually go into App Engine APIs. So you can see, let me actually close this panel so that we can actually clearly see. So you can see App Engine. If I type it in, I can see App Engine Admin API. And if I go to App Engine Admin API for this specific project, let's see what would come up. You can see that the API is not enabled right now. One thing you can do is to enable this API in here. The other way to enable the API in here is if I go back and let's say I try to create a new App Engine application, the first thing that we are choosing is a region. So whenever we are creating an App Engine application, the first thing that you need to choose is, which region you'd want to deploy your application in? We already discussed the fact that within one project you can only have one App Engine application. You can have multiple services under the application, but there can only be one container, one application at the top. And right now you are also seeing the fact that that application in a specific App Engine is also associated with a single region. So if you want to create App Engine applications in multiple regions, you need to create multiple projects. So for now, let's choose the default, which is suggested and say create app. So this would create our App Engine application. And I can also choose the language and the environment. You can choose the default, which are suggested by App Engine, is suggesting Python and Standard, that's fine. Let's go and say next. And now, you can see that it says your app is being created and your App Engine app has been successfully created. Now, we want to deploy a service to the App Engine. How can we do that? The way you can do that is by going to Cloud Shell. So I'll go to Cloud Shell. And over here you'd see a very, very interesting option in here. So over here, you can also click open editor to open a editor. So if I click open editor over here, I'm getting an error saying it's unable to load code editor because third party cookies are disabled. What I would do is I would open it in a new window. So I'll say open it in a new window. So what you'd see is that a code editor would be opening up so that we can edit our project in there. I'm using Safari, and looks like I'm having a few problems in Safari with the code editor. Let's go to Chrome. I've already authenticated to Google Cloud before. That's the reason why it would open up the terminal directly without asking me for password or anything. And I can go in here and say open editor. Once you click editor, the editor would open up in half screen. I don't really like half screens, so I'll say open in new window. And this would take a little while. So let's wait for it to get started. When you open a Cloud Shell editor, you'd see that you'll be able to edit code in here and you also have the command prompt available if you want to run any commands. We'll use both of them right now. What I'll do in here is say open folder. And I'll choose my home folder and say open. Now over here in the home folder is where I would copy all the things that we have already in the downloads. If you look at the zip that you'd have downloaded earlier and extract it, you should see a folder called downloads. And in downloads, you should see multiple folders. The specific one which we are interested in right now is App Engine. So if you go into App-Engine, App-Engine, there are two services which are present: default service and my first service. What I would do now is I would drag and drop them in here. So I'm dragging them from here and putting them into the explorer part of it. You can see that I'm putting it into the explorer part of it. And you'd see that those files would be automatically copied in. And if I go into the command prompt, and if I say ls, I should be able to see default service, my first service. And you'd see that these folders have a few files as well. You don't really need to understand these services in depth. A high level overview should be more than sufficient. In each folder there are three files. One is app.yaml, which is the configuration for App Engine for that specific application. Main.py is where the Python code for this specific application is present. Requirements.text is the place where we are defining the dependencies of this specific application. If you're familiar with Maven, then this is very very similar to Maven pom.xml. If you're familiar with JavaScript then this requirements or text is very very similar to, is very, very similar to package.jason, where you define the dependencies for your node application. So if I open up the requirements or text, there is a framework defined, which is Flask, which is an framework which is a Python framework to develop simple web applications or REST API. And if you open up the main.pi, it's very, very, very simple code. There is nothing complex in here. All that we have is we have defined a default service which returns are simple response back, my default service v1. Now it's asking me for a preview of new Python language server to get. I'm not really worried about it, so I'll say close. So this is a very, very simple Python application which returns my default service v1. And you have an app.yaml where we have simple configuration present in here as well. We are going to use a run time of Python 3.9. App.yaml is the application configuration that you're providing to App Engine. When now you're trying to deploy this application to App Engine, it would look at this app.yaml file and it sees that you are configuring a run time of Python 3.9. And if you're deploying it is to App Engine standard, it would make use of the Python 3.9, Python 3.9. Let's first start with the default service. So we have two folders in here: default service and my first service. We'll talk about why we have two services a little later. For now, let's focus on just the default service. So over here I'll say CD default-service. So over here carefully see which folder I am in. So I'm in the default service folder. And over here I'd want to do a gcloud app deploy. We want to deploy this app to G Cloud. But before we deploy the app, one important thing that you need to always be focused on is which project you are deploying this app to, right? And we had two projects. Which app project are we making use of? Let's look at the

config setting. Config List. It's asking me credentials. Let's authorize. You can see that we don't really have any project configured. So let's start with configuring the project. So gcloud config set project, and we want the App Engine project. So let's go in App Engine project. It's a nice name, Meladic Furnace and 304906. Interesting. So let's set the project to that. So now we have set the project to that. You can see three things in here, right? Which idea making use of what is the current folder and the project as well? The active project is Meladic Furnace 304906. Let's now go ahead and say gcloud app deploy. The important thing is if I actually do a ls in here, you'd see that the app.yaml is present directly in here. So if I actually do a cut app.yaml, you would see the content of app.yaml right here. So it contains runtime Python 3.9. That looks good. So let's do a clear. Let's deploy the app. So gcloud app deploy. So let's go and try that. Cool. It's now asking me a few things. Let me make this a little bigger so that you can see more details. Gcloud app deploy, services to deploy. You can see that the descriptor which is being used is app.yaml. So to the gcloud app deploy command, you can also pass in what is YAML file containing the app configuration. However, the default is app.yaml. And because we have the same name, app.yaml, it uses the default configuration. And this is the descriptor. The source is present from this specific folder. The source of the project is default-service and it's showing the target project. And you can see that the target service. We talked about the fact that an application can contain multiple services. One of those services is called a default service. So if right now we did not specify a service name at all, that's why it's making use of the default service. And we did not also specify a version for this specific deploy. So a little later we'll see how to specify a version when you're deploying it. But for now, we did not a version, so a calculated version is made use of. Now, when I tried to deploy, I got an error. It says access to bucket denied. It says you must grant storage object to your permission to a specifics service account. Now, what we'll do is we'll fix this and then try and understand what's happening, right? So what can we do to fix this? What you can do is to go over and search for IAM. IAM is where you can configure permissions for different things. You can configure permissions to people and configure permissions to applications, identity and access management. Let's go over to IAM and admin. Over here in the members, you can see all the different members which are present. And if you look at this specific thing. So this is a specific name, and you can see that there is this specific member with that specific name. And what I'll do now is I'll click edit member. I'll need to give him a new role. So I'll go ahead and say add another role. And what is the name of the role? The role is also present in here: storage object viewer permission. So I'll go and say storage object viewer permission. And say save. Now, this role has both these permissions. And now I can go back and try and kick off our deploy, gcloud app deploy. Let's see if this would be successful this time. So do you wanna continue? Yes. Beginning deployment. It's saying uploading zero files because the cloud files were already uploaded, but failure happened with the build. And now it's saying updating service. Now let's try and discuss what was happening and why did we need to give additional roles. Whenever we are deploying something to App Engine standard, what happens in the background is a deployment package is created, and the deployment package is told to an object storage service, which is present in Google Cloud, which is called Google Cloud Storage. How the package is created using a continuous integration and continuous deployment tool, which is present in Google Cloud, which is called Cloud Build. Cloud Build needs to access the package which was uploaded to Google Cloud Storage. And the error we got was that cloud build was not able to access the package which is present in Google Cloud Storage. And the way we give permission to Cloud Build is by giving access to something called a Cloud Build Service Account. So there is a service account which is automatically created when we started the deployment. And we went to that service account and we gave it permission to access data which is present in cloud storage. So we gave it storage object viewer permissions. And within a little while, I see that the deployment was successful and the URL is printed in here. If I actually pick up the URL from here and go and execute it, you'd see my default service v1. Where is this coming from? We saw that earlier. This is where it's coming from. Main.py my default service v1. So this is where that text is coming from And this it's very, very, very long step. We were able to finally deploy our application to Google App Engine. Let's try and understand it a little more in the next step. I'll see you in the next step.

Instructor: Welcome back. In the previous step we created our first version of the service and we were able to successfully deploy a two app engine. Let's now go into the UI and see what's happening there. So let's go in into App Engine. So I'm in this same project learning app, Engine project I'm going to App Engine. I want to see what are the things we should visit in the app engine. So this is the app engine dashboard where you can look at some of the metrics around specific versions. You can see that this is the version. Right now there is no data present because we are just launch it up. If you give it a little time you should see some data come up in here. If you scroll down you can also see what is the billing status, what is the cost. And you can also see what is the current loan on this specific application. You can also see if there are any application errors, server errors and client errors. We don't really have a fair by con configured so don't really worry about it. Now let's go to services. Services is where you would see the default service. We have created one service, which is the default service and that's what you'd see in here. You can see that there is one version for this specific service. So if you go to versions, you'd be able to see the version which is serving the traffic right now. So you can see that a hundred percent of the traffic for that service is being served by

this version. And you can see if I go to instances. Now, if I go to instances and scroll down a little you can see the current instances which are running. Now you can also get the same information on the command line as well. So for computer it was GCloud Compute. For App Engine, it's GCloud app. So all commands start with app. Earlier to deploy the application we use GCloud app deploy. But for now, we want to list the services. So I would say GCloud app services list. So I'd want to list all the services which are present. You can see that there is one service which is present, default, and one versions in it. You can also list versions, so GCloud app versions list. So what are the different versions which are present? So you can see that there is one default version which is present and it's handling all the traffic. And you can also list the different instances that are part of the app. So GCloud app instances list. So this would list all the instances that are present right now. So that's good. We have one version of this service deployed. Now you don't want to deploy another version of this service. How can I do that? The way we can do that is by actually using v2. So my default service V2 is what we would want to use. And now I would want to deploy V2 of the new service. So V2 of the new service would become v2. How can I deploy V2 of the service? So I can say GCloud app deploy and then the version label would be automatically assigned by App Engine. Or I can take control as well. I can say, I would want to specify the version. So I can say, GCloud app deploy hyphen version is equal to v2. So I'm specifying the version, the new version as v2 and let's see what would happen. So it's trying to now deploy the V2 and it says, okay, app.yaml is where your configuration is. This is the target project, that's fine. The target services default, that's good. And the target version is v2. And there is also a URL which is presented here. So it says, do you want to continue and say yes? So let's go ahead and say yes and this would deploy the application. So it would upload the changed file alone to Google Cloud Storage. And it would start running the cloud to deploy the package to app service. So all that happens in the background without me worrying about it at all. But what is important is while deployment is in progress you'd see that this application V1 is still running. So you can keep sending a few requests to this. So you will see that this is still running. So even when we are deploying a new version, you'd see that the V1 of the application is still running. So let's wait for the deployment to complete. It should take a minute or so, not more than that. Okay. It shows. It says updating service done. Setting traffic split for service is done. And I'll try and request this once. What comes back? You can see that V2 is now live. So as soon as V2 is deployed, it switched traffic to v2. So that's what was done when this was updated. So setting traffic split for service default done. Right now we are just doing a deploy traffic is automatically switched over to the new version of the service. And if I actually try and access this now you can see that V2 is the one which is responding back. And now, what I can do is actually go and save GCloud app versions list. What do you expect? You can see that there are two versions. You can see the traffic split. You can see that V2 is serving all the traffic enough and the earlier version is not serving any traffic. However, both the instances are up and running. Now I want to find out the URL for this specific version. So I would want to be able to see if this version is still running. How can I find the URL? So you can say GCloud app browse, oops, it should not be browser, it should be actually GCloud app browse. Let's see if it works. You can see that it's giving me the link to the active version. I'd want the link to a specific version. So I can say GCloud app browse hyphen version is equal to. Let's go and specify the version from there. Let's go and pick up the version from here and paste it in. Oops, I don't need the double codes. Let's move the double codes. So version is equal to this or version space this. So you can see that there is a way you can access the old version as well. So when you deploy the new version the old version is still running. However, it is not serving any traffic. If I click that link. control and click that or command and click that you'd see that this is showing v1 back. If you look at these URLs the difference is over here, it's version dot the name of the service, dot a specific suffix. So that's how the URLs are built. How about the default URL? You would see that it just has the name of the service dot a specific suffix. In this step we deployed a couple of versions of our application and we were able to successfully do that. I'll see you in the next step.

Instructor: Welcome back. In the previous step, we deployed the V2 of the service and we saw that the traffic was automatically shifted to V2. Sometimes you won't really want that to happen automatically. You'd want to deploy the new version and slowly switch traffic to the new version. How do you do that? Let's look at that in this specific step. So earlier we did G Cloud app deploy and we specified the version V2. Now I would want to slowly control the migration of traffic to the new version. How can I do that? Before that, let's create a new version. So let's call this V3. So I'll save this. So I'm changing the main.py to V3. And what I would do now is, I would go in here and now I'll go into the command prompt and I'll say, this is V3. However, I don't want to shift the traffic immediately to V3. We want to be live, I want to test V3 and then switch traffic to V3. And the option which is provided is no promote. So no promote is the option, hyphen, hyphen, no, hyphen, promote. This is the option where you can deploy the new version without switching traffic to it. So this traffic would still be going to the old version, which is V2, right? So let's go ahead and do this. G Cloud app deploy, hyphen version. You can see it shows all the details. Version is V3, that looks good, let's go ahead and say yes. The default is to promote. However, the option which we used is hyphen, hyphen, no promote, which means it will not be promoted. So a lot of these yes and no options in Google Cloud would have very, very similar options. So hyphen, hyphen, promote, and the reverse of that is hyphen, hyphen, no

promote. I really like that because it makes it very easy to remember. Now let's see if the service is deploying fine. Yep, it's going in. So the service is getting updated right now. However, you'd see that after updating the service there is no switching of traffic. So if you go over to the URL, the default URL, the service URL is still, what is it referring to? V2. However, what you can do is you can go and say, I don't want to actually test V3. So you can go and say add browser version V3 and create the URL for it. Actually you can say, G Cloud add browser version V3 and take that URL and you can test the V3. Hey, I got the right response back so my V3 is working. Basically you would have your users test it and then you can go and say, I want to make it live. But I don't want to make all the traffic go to V3 directly. I'd want to actually switch some of the traffic and observe what's happening in production. How can I do that? So I can say G Cloud app services. What I want to do is to set traffic and I would want to split the traffic. So split is equal to and I would want to specify this split. The way I would want to split it is I want send V3 only 0.5. Probably if it was a production application, I would probably send 0.5 or 0.1. Over here, I just split the traffic between V3 and V2. So half the traffic to V3, half the traffic to V2. And I can say, coma, V2 is equal to 0.5. So half the traffic to V2, half the traffic to V3, that's what we would want. Let's recenter, oops, I had a typo. It should not be split. It should be splits. So G Cloud a traffic set traffic splits, S P L I T S, and it should be specified with the hyphen, hyphen. So hyphen, hyphen splits is equal to this. Okay, now it (indistinct) traffic. So for V3 0.5, V2 0.5, and it says splitting traffic by IP. So the default way to split traffic is by IP address. So depending on where the traffic is coming from, it would be split. So let's go ahead and say yes. So now I'll take this URL and what I would do in here is cURL. So I'll do a Watch cURL this URL. So cURL this URL will send a get request to this URL. Watch cURL would actually send a get request to this URL every two seconds. So let's see what would happen. Hmm, V2, V2, V2, V2, V2. So over here I'm getting V2. You might have gotten cURL, you might even get V3. But the important thing is because I'm sending this request from the same IP address and I've decided to split by IP address, once a request comes from my IP address, it would serve the same version. And that's not good. Let's do a control-C or command-C. And let's actually change how we set traffic. Splitting by IP might be useful in a lot of scenarios, but in the scenario that we are in right now where I'm sending all the requests from a single IP, I want to split by a different option. So I can say split by random. So randomly split between the two versions, send half the traffic to V3 and V2. So splitting traffic by random, that's cool, let's go and say yes. And you'd see that the traffic change is immediate. So it's sitting the traffic split. And if I actually start Watch cURL again. So I'm executing the watch cURL that URL again. You can see now V2, V3. So it would actually randomly split traffic between V2 and V3. There is one other option to split traffic by. You can also use a cookie. So your application can set a cookie with a specific name, with the specific version you'd want to serve the request from. Now I'll do a control-C or command-C to break this. In this specific step, we looked at how you can actually control the traffic migration from one version to another. We deployed V3 without migrating any traffic to exit. After that, we sent half the traffic to V3 and now we are happy that all the traffic to V3 is looking good. So I don't really want to actually split traffic. So I want to send all traffic to just V3. You can go in here and say V3 is equal to one. So right now what we are doing is we are setting all traffic. Please go to V3 and you can press enter. So now I can say yes. So this would send all traffic to V3. And now if I send watch cURL requests, you'd see that for some time for about two or three minutes, I saw that the traffic was going towards V3 and V2, so making V3 the only version live took about a couple of minutes, and after that I was seeing all the requests were sold from V3 only. So this update took a little while but at the end we just have V3 live. I'm sure you're having an interesting time and I'll see you in the next step.

-: Welcome back! In the previous steps, we were playing with deploying multiple versions to our App Engine. If you go to the cloud console, and if you look at the services, you can see the services in here. And if you go to versions, you'd see that there are multiple versions, and you can see the traffic allocation in here. Some of the things you are doing from the console, you can also do it from the UI. So you can actually... Let's say I'd want to migrate traffic to a specific version, so if I want to switch all the traffic to V2, I can go to V2 and say, migrate traffic, then all the traffic would switch to just V2. Or if you'd want to change how we are splitting traffic, you can go in here, click this and click split traffic. And you can see the current configuration which is all traffic is going to V3 by IP address. If you want to make any changes, you can actually make those changes in here. You can add a version and configure how much that version should receive. So all the things that we are actually doing, right now, with respect to splitting the traffic and migrating the traffic, you can actually also do from the UI. Now, in addition to a default service you can actually create a number of other services. Let's see how to do that in this specific step. So, if I actually go back, "Cd.."... So, Cd to the parent, slash my first service. So, that's the thing I'm doing. So I'm cding into the "my first service" folder and if you look at the app.xml, app.yaml in here, we have specified a service name in here. So we would want to create a service with the name as "my first service". Look at where we are specifying the service name. The service name is specified in app.yaml and we are cding into that folder. And now, I'm actually doing a deploy again. So, GCloud app deploy. So the way you would actually deploy a service with a specific name is by configuring the service name in your app.yaml and calling GCloud app deploy. So if you look at the app.yaml in default service, it does not have the service name. That's why this is deployed as a default service. However, over

here, in this specific one, we would want to have a specific app name. You'd want to call this "my first service", microservice one, microservice two, microservice three, whatever. Let's say continue, yes, and it would deploy the application. So, it would start deploying to a new service. So what would happen now is a new service would be created and the first version of that service would be created. Because we did not specify an explicit version, it would give it a random version. So, if you look at the code for this it's very similar to the default service. The only thing that we changed is this would return "my first service V1". Cool! The service is now deployed. And you'd be able to access that using this specific URL. So, I can actually copy this specific URL. So, myfirstservice.meledicfurnace.uc.r.appspot.com. Let's copy this, paste it in here, and my first service V1. And now, if I go to services... And let's refresh the page, it's still showing just one. Come on, show me two service now. You can see that there are two services. Similar to this, you can actually create any number of services that you would want. Similar to how we created multiple versions in default, you can also create multiple versions in the my first service. Now let's try and run a few commands against my first service. Let's do a GCloud app services. What would it list? Oops, I have to actually add in a list. So, GCloud app services list. It would list the services which I present, default, three versions. my first service, one version. And I can say GCloud versions list as well. What it would list? It would list four versions. So, there are three from default, and there is one from my first service. That's cool. Now, let's say I would want to browse to my first service. How can I do that? GCloud app browse. Oops, actually it should not be browser, it should be browse. So GCloud app browse. What does it take me to? It would take me to the active version of the default service. However, I would want for a specific service. So, I would say "_service="... And over here I can say "my first service". So, from now on, whenever you'd want to refer to my first service, you need to specify "__service=my first service". And you can see that the URL for the my first service is very, very similar to the default service URL, except that it has "myfirstservice." before the project name and the suffix. Now, I can use this URL and browse through. You can also get a version specific URL. Now, if I would want to access a specific version then I can also add in a version. So, I can say "__version=" let's say... Oops, what's the version? Let's go and find the versions. So I can go to versions. And it's showing the versions for default service. Let's switch to my first service and you can actually type in this version. So I'll copy this, and I'll paste it in here. So this would give me the URL to a specific version. So you can see that the version is now attached to that specific URL. The idea behind the last few steps is to help you play with App Engine. We saw that you can create multiple services. Each of these services can have multiple versions, and if you don't specify a name for the service in the app.yaml, then it be deployed to the default service. And we also saw how the URLs are formed. App Engine is one of the most important topics as far as the exam goes, so what I would recommend you to do is to play around with App Engine a little bit more, and I'll see you in the next step.

09 - Google Cloud Kubernetes Engine for Cloud Architects
Welcome back. In this section, let's look at one of the most important services as far as this exam is concerned. Google Kubernetes Engine. We have talked about container orchestration earlier and Kubernetes is the most popular container orchestration tool, and Google Kubernetes Engine is the manage service offered by Google Cloud platform for Kubernetes. Let's get started with GKE or the Google Kubernetes Engine. Kubernetes is the most popular open source container orchestration solution. It provides cluster management. Basically, whatever workloads you'd want to deploy, you deploy them onto a cluster. Kubernetes allows you to create clusters and manage them including upgrades on the cluster. In a single cluster you can have different types of virtual machines. Different nodes in the cluster can have different hardware and software configurations. Kubernetes provides you with all the important features that you expect out of container orchestrator tools; autoscaling, service discovery, load balancing, self healing and zero downtime deployments. And what is the managed service provided by Google Cloud platform for Kubernetes? It's called GKE or the Google Kubernetes Engine. It's a managed Kubernetes service. With Google Kubernetes Engine, you can minimize your operations with auto repair. Whenever a node fails it would be automatically repaired and auto upgrade. You can have the entire cluster auto upgrade to keep using the latest version of Kubernetes always. Google Kubernetes Engine also provides Pod and Cluster Autoscaling. Why is this important? You might be running multiple microservices in a Kubernetes cluster and these microservices might be running on different nodes of the cluster. Pod Autoscaling deals with increasing the number of instances for a specific microservice. So, if I want to increase the number of instances of microservice A, I would need to increase the number of pods belonging to microservice A that are running. However, if I would want to increase the number of instances where microservice A runs, I would also need to increase the number of nodes in the Kubernetes cluster, and that's where we would need Cluster Autoscaling. As we keep increasing the number of instances of microservices, you need Port Autoscaling and Cluster Autoscaling as well. Google Kubernetes Engine integrates very well with cloud logging and cloud monitoring. You can easily enable them with very simple configuration. So, you can look at logs and the metrics around your Kubernetes cluster. Google Kubernetes Engine uses an operating system called Container Optimized OS. This is a hard end operating system optimized for running containers, which was built by Google. You can attach persistent disk and local SSD with the nodes that are part of the cluster. In this step, we got a 10,000 feet overview of Kubernetes and the Google Kubernetes Engine. In the next step, let's get our hands dirty. Let's start playing with Kubernetes.
-: Welcome back, are you excited to start playing with Kubernetes? Let's get started on a journey to deploy a simple microservice to Kubernetes. So, let's have some fun. Let's get on a journey with Kubernetes. Let's create a cluster, deploy a microservice and play with it in a number of steps. So, let's get started with this, step one, what is step one? Let's create a Kubernetes cluster with a default node pool with a set of nodes, you can either say, G-cloud container clusters create or we can use cloud console. Let's go to cloud control first. So I'm inside my first project, and what I would do is I would create a new project for Kubernetes. So I'll say my Kubernetes project and say create. So this would create a new project for us, and inside this we would want to create a Kubernetes cluster. So the project is created, let's go inside the project and search for Kubernetes engine. Kubernetes engine. Yep. That's what we are looking for. So before we are able to use the Kubernetes engine we need to enable the Kubernetes APIs. So that's the first thing that would happen now. You can see that it goes to the Kubernetes engine API. I can go ahead and enable them as it says in here. Kubernetes builds and manages container based applications powered by open source Kubernetes technology. So whenever we perform any operations with Kubernetes whether we are using console or whether we are using Cloud Shell in the background we are making calls to Kubernetes Engine API. The same is the case with everything that we did earlier. Compute Engine would be talking to compute engine API, App Engine would be talking to App Engine API. Inside the Google Cloud platform there are a number of APIs like this and whenever we want to use a specific service we need to first enable the APIs for that. The enabling of APIs is taking a little bit of time. Let's wait for it to complete. You can wait for a few minutes and then go into Kubernetes engine. You can just type in Kubernetes engine and you can go to Kubernetes engine. You can go in here and say Create cluster. When you click create cluster, you should see a popup like this. Select the cluster mode that you want to use. The options that you have are standard and autopilot. In standard, you take complete ownership of the cluster. In autopilot, you delegate cluster management completely to GKE. Let's quickly look at what autopilot is all about. The autopilot mode is actually a new mode of operation for GKE. Earlier, the autopilot mode was not present. The only mode which was present was standard. This is where you can say, I want 5 notes, 10 notes, 15 notes. So you are responsible for managing the complete cluster but with autopilot you don't need to worry about it. The goal of the autopilot mode is to reduce your operational cost in running Kubernetes clusters. It provides you with a hands off experience. You don't really need to worry about managing the cluster infrastructure like nodes or node pools. The GKE would completely manage the cluster for you. However, for now, let's start with standard as the configuration. So let's click configure over here, right beside standard, let's go ahead and configure nodes and everything. You can do the same thing by using the Command G-cloud container clusters create, important thing to remember is earlier

we were talking about G-cloud app, App engine G-cloud compute for compute engine. And now, when we are playing with Kubernetes, it's G Cloud container. So you don't have to create a cluster for container. Important thing to remember and observe is that when we are creating a Kubernetes cluster the command name starts with G-Cloud container. It says new cluster creation experience in here. This screen changes very, very often. So this cluster creation experience changes a lot so don't really worry about it. Whatever defaults it offers you, just take them. The only thing I would need to change is the name of the cluster. I'll call this my hyphen cluster. You can create a zonal cluster or regional cluster. I'm okay with creating just a zonal cluster and I'll take the defaults for most of the other things. And I would just say create. In this step, we got started with creating the Kubernetes cluster. The creation of the Kubernetes cluster will take a while. I'll see you on the next step.
Welcome back. The creation of the cluster took about five minutes, and after that, I can see that my cluster is now up and ready. You can see that by default, the cluster is created with three nodes and each of these nodes have two vCPUs each, so total of six vCPUs, and we have total of 12GB memory. So we would want to now deploy microservices to this specific cluster. What we can do is to connect to this cluster from our cloud shell. So, let's go ahead and say reconnect, and let's get the project ID for this specific project so that we can actually set the project ID. So this is the Project ID, (indistinct) images project. Let's go into Cloud Shell, and say Check cloud config, set project, and the project id. So this would be the default project from now on. let's do authorize. The next thing we would want to do is to connect to this cluster. So we have created a Kubernetes cluster and we would want to deploy microservices to it deploy containers to it. The first thing that I would need to do is to connect to it. How can I connect to it? You can get the command to connect to it by clicking by clicking this icon in here and say connect. So this gives you the command as far as the exam is concerned. It's very important that you remember the command. So it's gcloud container clusters. So this is the same starting point to creating a cluster. When we are creating a cluster. Instead of get credentials, we'll be using create. However, now we would want to get the credentials of the cluster, which cluster? We would want to get the credentials of my cluster. You are specifying the zone and the project as well. So what I can do in here is copy this and go and paste it in. So G Cloud container clusters get credentials. My cluster zone and project that's the command and you can see that it fetches cluster endpoint and authorization data. And now the kubeconfig entry is generated for my cluster. We'll be using something called kubectl to run commands against the Kubernetes cluster. And when we run commands, we'll be checking at kubeconfig for the cluster configuration and then executing commands against that particular cluster. So what we are doing right now is step two which is log into cloud shell. That's done. Step three, which is to connect to the Kubernetes cluster. And the way we connect to the Kubernetes cluster is executing the command gcloud container clusters get-credentials so that's the command which we have just executed. Now I would want to deploy a microservice onto this specific cluster. Now let's try and deploy the microservice to Kubernetes. And to deploy microservice to Kubernetes, we need to create something called a deployment and a service and we'll be using something called kubectl. So the command is something of this kind: kubectl create deployment, the name of the deployment and what image do you want to deploy? What we have done is we have already created this docker image for you. This docker image is already pushed to docker hub. A little later, in a separate step, you will see how this image was created. For now, the important thing for you to remember is that this dock image is available in docker hub and we are going to pick it up from there and use it in Kubernetes. And the complete command would be something of this kind. So I'll try and type this command. It's kubectl create deployment. So we'd want to deploy a microservice, the name is hello world REST API and I will need to provide what is the image -: - image=in28min is my docker id. And this specific microservice or the rest APIs Image name is hello world REST API. And the image tag that we want to use is 0.0.1.release. What I would recommend you to do is to actually take it from the presentation and run it as it is. You don't really want to make a typo in this. So kubectl create deployment hello world rest API. image is called in28min. Hello world rest API :0.0.1.release. Now one important thing that you need to remember is until now we had been using gcloud container command. If you wanna create a cluster gcloud container clusters if you want to get the credentials of a cluster gcloud container clusters get-credentials. So if you want to directly play with the cluster if you want to increase the number of nodes in a cluster if you want to add a node pool to the cluster what you would need to do is to use the gcloud container clusters command. However, if you want to deploy something to the cluster so I would want to deploy a microservice to the cluster I would want to expose the microservice to an external world. In those kind of situations, you'd use a kubectl command. So kubectl is a Kubernetes specific command line whether you have a Kubernetes cluster deployed in AWS or Azure or Google Cloud or even on your local machine or even in your data center, wherever Kubernetes is present. If you want to deploy a microservice, you can use kubectl. So gcloud is Google cloud specific thing to create the clusters. To create the cluster we would go to Google Cloud for things. However, to deploy something to the cluster we would go with kubectl, which is a which is a cloud neutral solution. Now once we created a deployment you can look at the deployment details you can say kubectl get deployment and you can see that the hello world rest API one one. So one instance is ready and it's up to date and it's available and it's been created almost two minutes back. Now I would want to access whatever is inside this

particular deployment. How can I do that? You need to expose this deployment to the outside world. So the command is kubectl expose deployment hello world rest API. We want to expose it using a load balancer. So type is equal to load balancer -- port = 8080. So kubectl expose deployment hello world rest API -: -type = load balancer -- port = 8080. The container runs on port 8080, so we are exposing port 8080. What is internally happening is when you expose a deployment something called a Kubernetes service is getting created. So we can look at the status of the Kubernetes service by saying Kubernetes get service or services. So you can use service or services without a problem. You can see that kubernetes get services would get me the fact that there is a service which is being exposed with the type load balancer. And you can see that the external IP for it right now is pending. So there is a default service which is always running. Whenever we create a cluster, which is Kubernetes we don't really need to worry about that one. What we are interested in is the hello world rest API service that we have just created. So when we expose a deployment what we are creating is a service. So you create a deployment to deploy a microservice you create a service to expose the deployment to the outside world. And the type of service that we are creating in here is a load balancer. And what we are waiting for is the load balancer external IP to be assigned. So I can say kubectl get services --watch and watch the progress. So you can keep looking at what's happening in here. So I can see that the external IP is now assigned, so that's cool. Now I can do a Control-C. You need to press Control-C to terminate the watch. Now I can send a curl request to this so I can take curl to this specific request :8080. So let's see what would happen. So we are sending a get request to that URL. You can see that it returns back healthy through and I can say :8080 slash hello world hello hyphen world and it's written back, Hello world v1. Now a simpler way to actually do that would have been to run it in the browser so I can actually pick up the URL and run it in the browser. Let's fix this. So the IP address of the load balancer colon port on which we are running it slash hello slash world. And you can see that hello world V1 is returned back. Our microservice, or not really a microservice Hello World microservice I would call this which is exposed out to the external world. In this step, we created a deployment and then we created a service by exposing a deployment and we saw that we were able to access it at a specific URL. We saw that the service was making use of a load balance at the type of the service that we created was a load balancer. Earlier in the course we created a load balancer directly. However, in here when we created a service in the background Kubernetes creates the global balancer for us. Let's look at all these details in the next step.

Welcome back. Let's reach over to console and let's pick up my Kubernetes project. That's the project where we created the Kubernetes cluster. And let's go to Kubernetes engine and that's where you'd be able to see your cluster. And over here, let's see a few details. I'll go into the cluster. Once you go into the cluster you'd be able to see the details of the cluster. You can see the name, what type of cluster it is. Whenever we're talking about Kubernetes, we create a cluster. As part of the cluster there are master nodes and worker nodes or just nodes. So you can see where the master nodes is running it and where the default nodes are running from. And you can see the version and you can see the size of the and you can see the size of the cluster three nodes right now. If you go to the nodes you can see all the nodes which are present in here. You can see that we created three nodes and these three node are created as part of one pool. A Kubernetes cluster can have multiple node pools. When we create the default Kubernetes cluster it creates one node pool with three nodes. However, if I would want, let's say specific kind of nodes let's say if I would want to create a set of node with GPUs. If I would want, I can go ahead and add node pools in here. So you can say add node pool, and you can say I would want to create a new pool. You can specify a size for that specific thing and you can configure everything related to that. So you can go to nodes and you can say I would want to have specific type of node. So you can configure what image you'd want to use. You can configure what type of node. Let's say you would want to run something which needs a GPU. You can go for GPU machine family. I'm not going to create this node pool but it's important for you to be aware that if you have specific workloads which need specific type of nodes, you can add node pools to your Kubernetes clusters. By default, there is one node pool, which is created but you can always add more. If you go to logs you can see the logs from the Kubernetes cluster. Now, if you want to look at what is running as part of the cluster, that's basically the workloads. So you can go to workloads. So the workloads is where you can see our deployment which we have created. We have created a deployment and we have one instance of it running. And each instance, which is running, is called a port. In Kubernetes terminology, each instance which runs as part of a deployment is called a port. If I increase the number of instances of deployment to three, then you'll have three ports for that deployment. If you click this and go further inside, you'd be able to see details of that specific deployment. You can see how much CPU it's making use of, how much memory, how much disc. You can see that all the monitoring and logging are on by default. So you can see how many revisions are present, for that specific thing, we made just one deployment for that. And you can see that there is one pod which is running as part of that deployment. And you can also see that there is an exposed service. So there is a service which is exposed. So hello world rest API. It's of the type load balancer and you can see the URL for it in here as well. So it's, if you click this, you'd be able to see that, Oops I want to go to that page. Please send me there. Yep. You can see healthy through, true coming back. And if you type /hello world, oops /hello world

you'll be able to see a response come back. You can always edit the deployment from the UI, but the best way, the recommended way of editing a deployment is through Command Prompt. You can go in here and see the details regarding the specific deployment. You can see the revision history for it. You can see what are the events, which are you can see the events which were related to that specific deployment. You can also see events by saying, kubectl get events. So this also gets you a list of events around what's happening with your specific cluster. You can see the last few things in here are matching whatever is present in here. So you can see that it says scaled up the specific rest API to one instances. And you can go to the logs and see the logs related to that specific deployment. One interesting thing is in addition to using commands to deploy you can also use YAML to deploy. Earlier we used a command to create the deployment. However, you can also use this YAML and deploy this specific service to Kubernetes. We'll take a deeper look at YAML a little later. If you go to Service and Ingress, this is where you can see this service which we have created. So we have created a service which is an external load balancer, Hello world rest API and the endpoint. One interesting concept in here is something called an Ingress. What is Ingress? What is Service? Service are set of bots with a network endpoint that can be used for discovery and load balancing. So over here what we are doing is we are exposing a deployment to the outside world using this service. On the other hand, Ingress are collection of rules for routing external HTTPS traffic to services. So if you have a number of internal services you can create an Ingress and redirect traffic to those services. So if you have a microservices architecture instead of creating individual load balances for each of the microservices, you can create just one Ingress. We'll also talk about Ingress and Services a little later as well. In this step, we did a quick walkthrough of the cloud control of Kubernetes engine. I'll see you in the next step.

-: Welcome back, in the last few steps, we made a lot of progress with our microservice journey. We started with creating a Kubernetes cluster. We logged into Cloud Shell. We connected to the Kubernetes cluster. Until here, we were using gcloud container clusters commands. And after that, we started deploying our microservice to Kubernetes. We created a deployment end service using kubectl, so we created a deployment and exposed it. And when we exposed a deployment, our service was created. Let's now see how you can actually increase the number of instances of your microservice. How can you do that? Do you think it would be difficult? The answer is it's very, very easy. So if I would want multiple instances for my deployment, all that I need to do is say kubectl scale deployment and the name of the deployment, Hello World REST API, and I'll need to specify --replicas is equal to, let's say, I would want three instances. You can see that it says it's scaled, and you can say kubectl get deployment. You can see that there are three available right now. And now I would go to the service, and I would refresh. When I refresh, I can see that the requests are coming from different instances. Let's actually take this URL, and I'll go back to Cloud Shell and say watch curl and the URL. Let's see what would happen. So you can see that it's coming in from CTB5Q observed for that. And now it's changing to 5DF and CTB5Q, CTB5Q. It's 5DFWB. You can see that the last few letters of the response is changing. What's happening there? If you do a kubectl get deployment, you can see that there are three instances, and each of these instances is called a pod. So if you do a kubectl get pod or get pods, you can see all the pods which are part of that specific deployment. You can see Hello World REST API and these five letters. The microservice which we have written will actually pick up the pod name and it'll return it back. And that's the reason why you'd see that when I actually refresh this, you'd see that the load is now being distributed among the different pod which are present in here. So this is SZXVW, and that's the one which is sending a response back in here. Maybe if I actually execute the request after a few times. If you keep executing the request, you'd see that you'd also get responses back similar to this and this. The important thing that you need to observe is the fact that we were not only able to scale the deployment up but the Kubernetes service which we have created is automatically doing load balancing between all active instances. Now, you can actually try to scale it up to, let's say, two instances, or three instances, or four instances, you would see that everything would be automatically created. So Kubernetes provides scaling and load balancing in a very, very simple way. Now, we saw how to increase the number of instances of our microservice. Let's say, I want to run hundred instances of this microservice or thousand instances of this microservice. Do you think we'll be able to run it? We have only created three nodes. So the default cluster that we have created has only three nodes. If I would want to create more instances of this deployment beyond a limit, I would need to actually scale up my cluster. So I would want to actually increase the number of nodes in my cluster. How can I do that? The way I can do that is by gcloud container clusters resize, and you need to give the name of the cluster, and you need to give node pull and the number of nodes. So what I'll do is I'll try and execute that command right now. So gcloud, remember we are going back to the cluster. That's why we're going back to gcloud, gcloud container and gcloud command all use clusters, instances, services. So gcloud container clusters resize. Next, you need the name of the cluster. The name of the cluster is my-cluster. Which cluster do you want to resize? It's my-cluster. And after that, whenever we're resizing, we're resizing a node pool inside a cluster. We're not really resizing the cluster directly but we'd want to resize a node pool. Like you might have multiple node pools so you need to specific which node pool you'd want to increase the number of nodes for. So I'll go to nodes, and you can see that the name of the node pool is default-pool. So resize cluster --node-pool. What do you want to

do? It's default-pool. So default-pool. I would want to resize this specific cluster. How many nodes do I want? We already have three nodes running. I don't really want to increase the number of nodes. So what I'll do is actually decrease the number of nodes. So gcloud container clusters resize my-cluster, I'm specifying the node pool and the number of nodes as two. So what is the clustered zone? Let's go back to clusters. Let's go back to the details. We have created a zonal cluster. So we have the cluster running in this specific zone, us-central1-c. So I'll go back in here. So I'll go ahead and add in --zone is equal to us-central1-c. Let's see if this would work. You can see that it says, "Pool, default pool for my cluster will be resized to two. Do you want to continue?" Yes, I would want to. So it's resizing the cluster now down to two. You can increase it or you can decrease it. And this is manual, right? So we are deciding what should be the size. The same thing with the pod earlier. Earlier, we specified a specific size. So this is called manual scaling. And manual scaling of the cluster is done by resize. And the manual scaling of a pod is done by doing a scale deployment using kubectl. What we're looking at are some of the most important concepts of Kubernetes and that's the reason why we designed this journey in such a way that we can go through the entire journey step by step. And let's wait for the cluster to be resized. The value shown below will be updated once the operation finishes. So we're still waiting for the operation to complete. The resize of the node pool is taking a long time, and what we'll do is we'll take a break. One of the important things to remember is that we're not really happy, right? We're manually increasing the number of instances and nodes. What we would want to do is to autoscale. Let's see how to do that in the next step.

Welcome back. The resize of the cluster took a long time but finally it was done. It took about 10 to 15 minutes before it was done. And now, through Cloud Shell and Cloud Console I'll be able to see that my cluster has just two nodes, so let it refresh. Yep. You can see that there at just two nodes. And as we talked about last step, we are not really happy about the fact that we are manually increasing the number of instances and nodes. That's where you can go for auto scaling. You can set up auto scaling for your microservice and for your cluster. How do we set up auto scaling for our microservice? Again, it's a very, very simple command and since we are playing with the microservice or the rest API we would need to use kubectl. So let's go in and say kubectl autoscale deployment. So we don't want to autoscale deployment, which deployment. Hello, world rest API deployment. And what do we want to do? We don't want to have a maximum of let's say four instances. So I would want to have a maximum of four instances. How do I decide about auto scaling? I would recommend to do it based on CPU percentage. So CPU percent is equal to let's say 70. So you can go up to a maximum of four instances and you can scale up and scale on, try and achieve a CPU utilization of 70%. Let's press enter. Cool. The kubectl autoscale deployment hello world rest API half and half max is equal four and CPU percent is equal to 70. And how does this auto scaling work internally? When I execute this command kubectl autoscale command, what internally gets created inside Kubernetes is something called a horizontal port auto scaling configuration. This is also called HPA, and if you actually do a kubectl get HPA, you should see something of this kind. You can see that the name is this and it is referring to this deployment. And this is the target. The target is 70% right now this utilization is unknown and the minimum number of parts is one. The the maximum number of parts is four and the current existing number of replicas is three. So this horizontal port autoscaler is what is created when we execute autoscale deployment. And this is how we scale our microservice. Now auto scaling microservice is good, but you'd want to also autoscale your Kubernetes cluster. The microservice deployments can only scale up to the level there are nodes in the Kubernetes cluster. Beyond the level, you might not have sufficient node and then you have to autoscale or increase the number of nodes in your Kubernetes cluster. How can you do auto scaling on a Kubernetes cluster? That can be done by using a command called G Cloud container clusters update. You'd want to update a specific cluster and you'd want to enable auto scaling. And you can set what is the minimum number of nodes and the maximum number of nodes. So you can autoscale both your applications and as your applications or microservices keep auto scaling they need more resources and to get more resources you can autoscale your kubernetes cluster as well. In addition to autoscaling, let's also look at configuration in this specific step. Let's say you have a microservice and you want to configure a connection to a database or something like that. Where do you configure that in kubernetes? The place where you would configure is something called a config map. So you can create a config map and set the values into a config map and you can have your microservice or your deployment pick up the values from the config map. The command is kubectl create config map and you can give it a name. So whatever name you'd want to give to the config map and you can specify the values. So you can specify the DB name, something of this kind. So let's go ahead and try that. So kubectl create config map. You can give it a better name. The application config might not be the right one in here. Let's probably call it Hello World Config. So I'd want to store the configuration for Hello World Config and I would want to store the database name in there. So that's it. You have created a config map and you can use this config map from your microservice. So kubectl get config map would get the details inside the config map. So you can see that there is a config map called Hello World Config. kubectl config map, Hello World Config. And this would get, and I would go ahead and kubectl describe config map. Hello world config. And then you can see the data inside that config map. So over here you can see the name is this and you can see all the data which is

presenting here. You can see that RDS DB name is present with the value of to-dos. In addition to configuration, you might also want to store secrets. Let's say you don't want to store database password. You can also store that in kubernetes as a secret. So instead of creating a config map, what we would do is we would create a secret. So I'll use this command. So it's kubectl create secret. The type of secret is generic, and this is a name we are giving it to it and we are saying these are the values that should be stored index secret. So let's try that. So let's do a clear and execute that command kubectl create secret generic. Let's just say hello, world secrets in here instead of to-do web application. So let's go in here and say hello world secrets. And we have the password in here. So now the secret is created so I can say kubectl get secret. And you can see that there is a default one which is already created by Kubernetes cluster by default. The one which we have created is in here Hello World Secret One, and let's say kubectl describe secret and the name that we have given it. Hello World Secrets One. What would we get back? You can see that you are getting IDs, password and you'll not be able to see the password in a plane text anymore because there's a secret. This would be encrypted and stored. So if you have a microservice and you'd want to store the configuration, you would use config map. If you want to store a password then you would use a secret. This is also called a secret map. In both config map and secret you can have any number of values. So you can store all the configuration that is needed by your microservice or even a set of microservices in a single config map and in a single secret map. In this step, we looked at autoscaling and configuration. Kubernetes makes it very very easy to do autoscaling as well as centralized configuration for your microservices. I'm sure you're having a wonderful time and I'll see you the next step.

Welcome back. Let's now go back to console. Let's go to web console and try and play with the workloads. So if you go to Workloads, and if you actually open up "Hello, World" REST API, and go into YAML. So this is where you can actually directly edit the YAML as well. So instead of actually editing the deployment using commands, we talked about the fact that we can edit the deployment YAML. Let's say I want only two replicas, so I can say I only want two replicas. And before I can change it, I need to click Edit. And now I can say two replicas. So instead of saying kubectl scale deployment or autoscale deployment, I can come in here, edit the YAML, and say this is my desired state. My desired state is that I want to have two replicas. I can configure that in a YAML, and I can say Save. And what would happen, in a little while, it says "Pod has warnings," because this is terminating. So you can see the third pod is terminating because we said we just need two. And if you actually wait for a little while, if I refresh this, you'd see that the number of pods would come down to two. So you can see that there are just two pods running at this particular point in time. So there are two ways you can actually deploy things to Kubernetes. One is by using commands, the other one is through YAML. Executing commands is called imperative style. Using YAML is considered to be a declarative style. So you can go in here and make changes to your YAML, and use that to deploy. If you have a YAML file, then even from the command line you can deploy it by using kubectl apply -f deployment.yaml So if you have a deployment.yaml file, then you can say kubectl apply -f deployment.yaml file, and deploy it. We have provided a couple of sample deployment files, for deployment and service, in the presentation. So what we are looking at right now is deployment YAML. This is a little bit of cleaned up version of whatever you're seeing on the browser. This is for a deployment, so you can see that kind is Deployment. If you want to increase the number of instances, you can change the replicas, let's say, to 5. If you want to change it to, let's say, 0.0.3 release, you want to change it to a different container release, you can directly change it in here, and then you can do kubectl apply -f and the name of the deployment file. Similar to this, you can also create a service using the deployment YAML. You can see that the kind in here is Service, and you can see that the port is configured in here. And also you can see that the type LoadBalancer is configured in here. You don't really need to understand the details of these deployment files. If you're able to map the command you have executed earlier to the deployment files, that's more than sufficient. Earlier, when we were running the command, we said kubectl create deployment. Number of replicas is 3. And we said, this is the image we would want to deploy. In the command we used 0.0.1 release. Over here it's 0.0.3 release. Similar to that is the service. We created a service using command kubectl expose deployment, and we specified a name for this service. We specified the port where it needs to be exposed on, and we said it's of the type LoadBalancer. So the same things that you can do through the commands, you can also do through deployment.yaml files. One of the most important things to remember is what we are doing in here is we are looking at a number of complex topics. Getting into Kubernetes world is not really easy. You have a number of things related to the clusters itself. Cluster, node pools, nodes. And there are a lot of things related to deployments. Deployment, service, ports. As far as the certification exam goes, it's fine if you understand things at a high level. In this step, we talked about deployment YAML file, which is a declarative option of deploying things to Kubernetes. I'll see you in the next step.

Welcome back. In the last few steps, we learned a lot of concepts and it's now time to get to the end of the journey that we have started. There are a lot more slides about Kubernetes that we will be talking about. Let's end the most important hands on paths here. Deploy a new microservice which needs nodes with a GPU attached. How can we do that? We discussed this, right? So you already have existing deployments and you would want to deploy new

microservices. How are these microservices who need nodes with GPU attached? How do you do that? You'd create a new node pool. You'd want to attach a new node pool with GPUs to our cluster. And the way we can do that, we saw how we can do that from the cloud console. You can also do it using GCloud. Important thing to remember, we are changing the node pool. We are changing the cluster. So we would use GCloud container. GCloud container node pools create the pool you'd want to create and in which cluster you'd want to create the pool in. In addition to this, you can also add a little bit of configuration on what kind of nodes you want to create as part of the pool. The command to list the load pools is GCloud container note pools list. So if you go in, find this in there. So let's go to the cloud shell. Oops, not here. Definitely not here. So let's try it again. GCloud container node pool list. It's saying, okay, tell me which zone, or which region. So let's start with listing the zone. Which zone is this cluster in? Let's go in. This is a new central one, C, so let's go there. You have central one C. And then it's asking cluster. Which cluster is it in? So let's do that as well. You can actually do a config set as well. So GCloud config set contains less cluster to the value and then that would be the default cluster. However, you can also specify cluster as part of your command. So which cluster? What's the name of our cluster? It's my-cluster. My-cluster. Let's get that done. So you can see that we have enough one default pool. If we want, we can add another pool by using GCloud container node pools create, give it a name, and say this is the cluster I would want attached to it. Now, once we created a node pool, what do we want to do? We need to tell Kubernetes that the microservice, the new microservice, which needs a GPU, needs to be deployed to the new node pool, not to the old node pool. How can you do that? The way you can do that is by setting up something called a node selector in the deployment.YAML file. In the deployment.YAML file, you can configure node selector and say cloud.google.com/gknodepool. So use this specific node pool. This is the pool which needs to be used to deploy this specific deployment. So to deploy a new microservice, which has specific needs, you can create a new node pool, and then in the deployment of the microservice, you can say make use of that node pool. Now we are ready to get to the end of our journey. How do we delete the microservices? You can first delete the service cube cuttle delete service and the name of the service, and then delete the deployment cube cuttle, delete deployment, and the name of the deployment. We are using cube cuttle to delete the service and deployment, and if you'd want to delete the cluster, it's cube cuttle, not cube cuttle, right? You're playing with the cluster, so GCloud. GGloud container clusters delete. And give the name and the details related to that specific cluster. We're not going to delete the microservices and cluster right now. I just wanted to mark the end of this journey. What we'll do is we'll play around with the clusters and the deployments a little bit more as we go further, and at the end of this section is when we would delete the Kubernetes cluster. Kubernetes was such a complex topic and we designed this journey so that you can actually have something to fall back on. You can refer to this journey whenever you have a question in the exam, and it would make it easy for you. I'm sure you're having a wonderful time and I'll see you in the next step.

Welcome back. In the last few steps, we learned a lot about Kubernetes. Kubernetes is really vast. We have a course of around 13 apps just discussing Kubernetes in depth in various clouds, AWS, Azure, and GCP. Trying to discuss Kubernetes in a few hours is a very, very tough task and that's what we are trying to do as part of this certification course. This certification is not just about Kubernetes, but everything related to GCP, so we are trying to cover the most important concepts from the perspective of an examination. The next few steps, let's review some of the important things that we have learned as part of our Kubernetes journey. The first thing that we'll talk about is the cluster. Cluster is where you run your workloads. Whenever you would want to deploy something in Kubernetes, you'd want to first create a cluster, and a cluster is nothing but a group of Compute Engine instances. There is a master node which manages the cluster and there are worker node where you run your workloads. Inside your master node, this is also called the control plane because this is what is managing everything. Whenever we execute a kubectl command, where are we sending the command to? We are sending it to the master node. And the master node would do whatever is necessary to execute that command. It manages the cluster and it says, Hey, I need a new node. Or the master node says, Hey, deployment, I would need a new instance for you. So the master node is the one which manages the cluster and the worker nodes are where our workloads or our applications or our microservices run. And as part of our master node, there are a number of components. The first one is the API server. There is a lot of communication which happens within a Kubernetes cluster and there is a lot of communication that happens from outside to the Kubernetes cluster. I'm executing a command, which part of the master node receives the request for a command? That's what is called an API server. This manages all communication for a Kubernetes cluster from outside and the communication from master to the worker nodes. The next one is the scheduler. When I say I would want to create a deployment with three parts, what does the master node need to decide? Let's say there are 10 worker nodes. It needs to decide which is the worker node, where instances of this deployment need to run, and the component in the master node which does that is called the scheduler. Is it sufficient if you just deploy things? You'd also need to manage if they're healthy. If they're not healthy, you'd need to actually replace them, and that's why you have a control manager. The control manager managers deployment and (unknown)assets. A little later we'll discuss what is the difference between a deployment and (unknown) asset. The fourth important master or component is E T

C D. It is a distributed database storing the state of the cluster. You need to store the state of the cluster somewhere. That's how you can provide high availability. And the database which is used is E T C D. This is a distributed database. We looked at the different components that are present as part of the master nodes. Now, what happens within the worker nodes? what are the worker node components? The worker node components are where our workloads run, where are deployments, and deployments are made up of instances which have nothing but ports. So the worker nodes are where our ports run. And in the worker nodes, there is a specific component related to Kubernetes, which is called kubelet. The worker nodes needs to talk to the master node, and that's what kubelet does. It manages communication with the master node. When it comes to Gt, there are a few different types of clusters. What are the different types of clusters? The first one is zonal cluster. So this is single zone. So there is just one control plane. There is just one master node, and all the other nodes are also run in the same zone. In the example that we used, we created a zonal cluster. We saw that the control plane the master node ran inside the same zone and the node were also running in the same zone. There is also a multi-zonal possibility for this zonal cluster. Over here, you have single control plane. Basically, your control plane is inside one zone, but nodes are running within multiple zones. There is also a regional cluster. In a regional cluster, replicas of the control plane run in multiple zones of a given region. So you'll run multiple instances even of your master node. And wherever your master node runs in those zones, you also have worker nodes. Why do you go for a regional cluster? Why do you want master nodes to be deployed to multiple zones? Because even the master nodes can fail. If there's a failure in the master node, you don't want to lose the entire cluster. You can have high availability by even distributing your master nodes. There is also something called a private cluster which is specific to a virtual private cloud. Virtual private cloud is nothing but a internal network that you can create in Google Cloud. A private cluster is something which lives within a VPC. And there are also clusters called alpha clusters. These alpha clusters are created with early feature APIs, which are nothing but alpha APIs. So whenever you want to test the brand new Kubernetes features, you can try alpha clusters. In this step, we looked at what is a cluster what are the components that are part of a cluster and what are the different types of clusters. We talked about worker nodes, we talked about master nodes and we talked about zonal, regional, private, and alpha clusters. I'm sure you're having a wonderful time and I'll see you in the next step.
Welcome back. In this step, let's talk about pods. What is pod? Pod is the smallest deployable unit in Kubernetes. A pod contains one or more containers. Each pod is assigned an ephemeral IP address. Earlier, we deployed our application to Kubernetes and if I say Kubernetes get pods... Oops. It should be kubectl get pods. If you get an error, make sure that you actually first connect to the Kubernetes cluster. You can get the command by actually going to the cluster. Over here, you can get the connect command. Run that, so kubectl get pods. You can see that there are two pods which are running right now. So a pod is where our microservices run. In a single pods, you can actually have multiple containers. Most of the pods, however, will contain just one container. And when we create a deployment with three instances, we'll have three pods. Over here, our deployment has two instances. If I say kubectl get deployment, how many instances does it have? It has two instances, and that's why it has two pods. If I do kubectl get pods -o wide, you can see more details about a pod. You can see that each pod has a IP address of its own. This IP address is a ephemeral IP address. If you kill this pod and create a new one, the IP address would change. If you have multiple containers in the same pod, then they all share network, storage, IP address, ports, and also if you are attaching any volumes or shared persistent disks with this specific pod. So if you have a deployment and you are creating a volume for the deployment, your deployment instances that are nothing but the pods can also access your volumes, and all the containers which are part of the pod can access all the things that are part of a pod. The pod can be in different statuses, running, pending, succeeded, failed, or unknown. Running is when a pod is running. Pending is when you are waiting for a pod to be deployed to one of the node. Succeeded is when its job is done. Fail is when a pod was not started successfully. Unknown is master is unable to find out what is the status of a pod. In this quick step, we looked at pod. Pod is the smallest deployable unit in Kubernetes. Each pod has individual IP address and a pod can contain more than one containers. Each instance of deployment is nothing but a pod. Let's talk more about deployments and replica sets in the next step.
Welcome back. In this step, let's talk about deployment and ReplicaSet. We have been talking about deployment a lot. Now, why do we need ReplicaSet? A deployment is created for each microservice. So, kubectl create deployment, and instead of M1, we were using Hello, World REST API. And we specified the image. We said, this is the Hello, World REST API 0.0.1 release image that we would want to deploy. Similar to that, if I would want to deploy a microservice, I can say M1, and probably put in the image for the microservice. A single deployment represents a microservice with all its releases. So you might have 2, 3, 5, 10 versions of your microservice. A deployment represents all versions of a microservice. One of the most important roles of a deployment is to manage new releases, ensuring zero downtime. Let's go in here. Let's do a clear and say, kubectl set image deployment. Which deployment? Hello, World REST API. And inside the Hello, World REST API, we would want to update the image. Which container do you want to update the image for? We'd want to update the container named Hello, World REST API. And we'd want to update the image to what? In 28 min Hello, World REST API :0.0.1,

0.0.2.release. So we earlier created a deployment. Now what we are doing is we are updating that deployment. We are saying, update the image of the Hello, World REST API to this. So this is the image. I would want to upgrade the release from 0.0.1 release to 0.0.2 release. I've already created this container and this already part of Docker Hub. So, I can go ahead and execute this command. This command might be tough to execute, so make sure that you pick it up from the backup of commands that we have provided you already. So let's go ahead and run this. You can see what happened. Deployment/apps, Hello, World, REST API image updated. Now, this is the Hello, World API that we released earlier. So, if I execute this, you can see that it's now returning Hello, World V2. How is it returning V2? Because now, we have automatically upgraded to V2. The image is updated to V2. So the same URL is now serving the V2 of the microservice. How can you get this URL? One of the ways you can do that is by saying, kubectl get services. So we had earlier created a service around this deployment, and this is the external IP. So you can use this :8080/helloworld. That's the URL that we used earlier as well. So the same deployment is now representing 0.0.1 release and 0.0.2 release. So a deployment manages new releases, ensuring new downtime. We were able to easily upgrade to a new release without any downtime. What is the role of ReplicaSet? ReplicaSet, on the other hand, ensures that a specific number of Pods are running for a specific microservice version. Earlier we were using V1, now we are using V2. Earlier, there was a ReplicaSet for V1, and now that ReplicaSet will be replaced by a V2 ReplicaSet. So a ReplicaSet represents specific version, specific deployment version, and its instances, and its Pods. Now, how can I find that out? I can say, kubectl get ReplicaSets. What are present in here? You can see two ReplicaSets. This ReplicaSet is the one belonging to old version. And for the old version, we don't have any Pods running right now. And for the new version, there are two Pods which are running. Now, what I would do is I would say, kubectl get Pods. And I would say, kubectl delete Pod. And I'll pick up this and delete it. So the Pod is deleted, and I'll do, kubectl get Pods. You can see that there is a new one automatically launched. So as soon as an existing Pod is deleted, the ReplicaSet says, "Hey, I want actually two instances to be running always. Somebody has killed a Pod, I would need to launch up another immediately." And what does the ReplicaSet do? It would ensure that at least two Pods are always running. So the duty of a ReplicaSet is ensuring that a specific number of Pods are always running for a specific microservice version. Deployment is responsible for multiple versions of your microservice. So deployment is only responsible for shifting from one release to another release. So it is making sure that there is no downtime when you are actually moving from one release to another release. Who is responsible for each specific release and ensuring that these specific number of instances are running? That is the ReplicaSet. So kubectl scale deployment M2--ReplicaSet equals to two. What we are updating is the ReplicaSet. We are telling the set, "Hey, you need to have two instances." Let's try that command. Kubectl scale deployment Hello, World REST API. Let's say I just want one instance. -: -replicas is equal to one. What would happen now? Let's do a kubectl get. Will there be a change in the deployment? Actually, the thing which would change is a ReplicaSet. So kubectl get ReplicaSets. So you can see that the design for the new ReplicaSet of V2 is now updated to one. So we only want one Pod to be running always. Even if one of the Pods is killed, ReplicaSet will launch a new one. If you deploy a new version of the microservice, it creates a new ReplicaSet. So if we update the image, that's what we have done earlier. Kubectl set image deployment. What happened? A new ReplicaSet was created. So a V2 ReplicaSet was created. The deployment updates the V1 ReplicaSet, and the V2 ReplicaSet based on the release strategies. So if you'd want to slowly move from one release to another release, the deployment can do that for you as well. These are called rolling deployments. In this quick step, we looked at what is the difference between a deployment and a ReplicaSet. I'm sure you are having an interesting time, and I'll see you in the next step. Welcome back. We have been talking about services for a long time. Let's quickly revise the most important things that you need to remember about a service. Why do we need a service? Each pod has its own IP address. How do you ensure that external users are not impacted when a pod fails and is replaced by the replica set? We killed a pod and it was immediately replaced by the replica set. However, our URL would still be working so our service URL will continue to work. Even though things change internally to the external users, there is no change. A new release happens and all existing parts of old release are replaced by ones of new release. We saw that happen, so we replaced all instances of V1 with V2 and even then we saw that the URL continued to work. So how do you ensure that external users are not impacted when there are changes internally? That's where we create a service. The service exposes your deployment to the outside world and also ensures that external users are not impacted even if something changes internally. We earlier created a service by using cube catalog exposed deployment. Half and half type is balancer half and half port is equal to 80 80 or whatever, whichever port you don't make use of. So expose ports to outside world using a stable IP address ensure that the external world does not get impacted as parts go down and come up. There are three types of Kubernetes services. The one which we created was load balancer. In addition, there is something called cluster IP. Sometimes you don't really need to expose services outside your cluster. There might be internal microservice that you are making use of. It's specific to whatever you are doing inside the cluster. You don't want it to be exposed outside your Kubernetes cluster. In those kind of situations, you can go for a cluster IP. It exposes service on a cluster internal IP address. A use case is when you want your

microservices only to be available inside a cluster intra cluster communication. The next one is what we have been using, load balancer. It exposes service externally using a cloud providers load balancer. Let's see what's happening here. So let's actually go to load balancers. Is there a load balancer created for us? So let's go to load balancing. You'd see that when we created a service, actually a load balancer was also created for us. This load balancer was created when we created a service. Internally we created a service with the type load balancer and externally a cloud provider's load balancer was created for us. This would work in any cloud. If you tried in AWS or Azure with the respective Kubernetes service, you'd see that a load balancer would be provisioned for you. You'd go for a load balancer if you'd want to create individual load balancers for each microservice. So for each deployment you are creating a separate load balancer. This might not be a good thing to do always because you don't want to use too many load balancers as well. The other option is node port. The node port exposes service on each node's IP at a static port. So there's a port called the node port and you are exposing the service at the node's IP address. A use case for going for node port is when let's say you don't want to create an external load balancer for each microservice. What you can do is you can expose all the services using node port and then you can create an ingress. You can create one ingress and you can actually route to all the microservices. Where did we see ingress earlier? If you go over to ingress when I type in ingress it's actually pointing me to services. So services kubernetes engine, that's where we need to go. And this is where you can see services and ingress. So this is where you can actually create an ingress. What you can do is you can expose your service as a node port. So if you have 10 microservices, you can expose all of them as node port and then you can create a ingress. So you can go in here and create an ingress. An ingress can route actually to load balancers and node ports. Particularly the use case you would use is you would expose the services as node ports and then you would have an ingress to expose them to the outside world. In this step, We talked about a Kubernetes service. You create a service so that external world does not need to know about what's happening inside your cluster. We talked about the three types of services; cluster IP, load balancer, and node port. A cluster IP will be useful if you were microservice or if the service you are exposing is used only within your cluster. A load balancer is used when you'd want to use a cloud provider's load balancer and expose your service to external users either on the internet or maybe within a specific network within your intranet. A node port can be used when you want to expose services. Add the node IP at a static port. One of the possibilities with node port is you can expose all your microservices as node port and you can have ingress, which can route to multiple node port services. I'm sure you're having a wonderful time and I'll see you in the next step.

Welcome back. In the earlier steps we talked about services. We talked about different types of services, and we also talked about a specific service type called load balancer, which enabled us to access our deployments. When we used the load balancer service type, we saw that a load balancer was created, and we were able to access the microservices which were deployed behind that specific service. There is another important concept called ingress. Ingress is nothing but a collection of rules for routing external HTPs traffic to US services. So you have deployments, services, and then you have ingress. What is the need for an ingress and how do we use it? That's what we'll be looking at in this specific step. Ingres is the recommended approach for providing external access to services in a cluster. So, you have a Kubernetes cluster and you have a number of deployments present there, and over the top of those deployments, you might have services that expose them to the outside world. And, if you want to provide external access to these services, ingress is recommended. Ingress provide load balancing and also they provide SSL termination. The great thing about an ingress is that you can control traffic by defining rules on the ingress resource. For example, you can say, if a request comes to this URL, I would want to send it to this particular service, and I would want to send it on this specific port. What you're seeing in here is the simple deployment configuration for a ingress. You can see that the kind for this is ingress. You don't really need to remember all the details about this. The important thing that you need to remember is that you can redirect requests based on the path to different backend services. So if you have 10, 15 microservices with one ingress, you can route to all those services. Earlier we saw that there are different types of services, load port, load balancer, and others. When we use the service type load balancer, an external load balancer was created for us, and we were able to access the deployment. However, the recommended approach to provide ingress is to use the service type as load port. Whenever you want to expose a service, use the service type as load port and expose it out using an ingress. Why is it recommended not to use load balancer service and why is it recommended that you use an ingress? Consider this scenario. Let's say I have 15 microservices or 20 microservices, if I'm using load balancer service to expose those deployments, what I would need to do is I would need to create 15 or 20 load balancers. Each time I create a load balancer service for each of the microservices, a new load balancer is created, and that's not good. When you use a ingress, this would create one load balancer which can redirect to multiple microservices. So, based on the path, you can redirect it to the appropriate microservice. So, if you're using a microservices architecture and if you are providing external access to users to use your microservices, ingress is the recommended approach to provide external access. Ingress allows you to use single load balancer and use it to control ingress into multiple microservices. I'm sure you're having a wonderful time, and I'll you in the next step.

Welcome back. In this step let's talk about Container Registry, or the image repository. What is Container Registry? You have created DACA images for your microservices. Where do you store them? You store them in Container Registry. Container Registry is a fully managed container registry provided by Google Cloud platform. An alternative would be to use Docker Hub. Docker Hub is a public Container Registry. You can integrate with CACD tools in GCP like cloud build and you can push images to the Container Registry. Container Registry also has features where you can secure your container images. You can analyze your container images for vulnerabilities and you can enforce a few policies. A typical image in a GCR of the Google container repository would have a tag like this GCR.io slash the project name slash the tag you give. So your microservice name, colon, maybe the tag version one, for example. In this quick step we looked at Google Container Registry. Google Container Registry is where you can deploy your container images to in Google Cloud platform. This is very, very similar to Docker Hub, however Google Container Registry is private by default. I'm sure you're having an interesting time and I'll see you the next step.

Welcome back. Until now, in this section, we make use of a number of pre-built Docker images. How do you create a Docker image and what are the best practices in creating Docker images? That's what we'll be looking at in this specific step. The idea behind this step is to give you a high level overview of how Docker images are created. You don't really need to understand all the details. A high level 10,000 feet overview of understanding how to create Docker images is more than sufficient. And that's what we will be doing in this specific step. Whenever you have an application and you want to build a Docker image for it, one of the possible approaches is to create a Docker file. So you can create a Docker file where you have all the instructions related to creating Docker images. What you're seeing in here is a Docker file for a simple node JS application. So the Docker file contains instructions to create Docker images. What we are seeing in here is this is the base image. So we want to make use of this image. We don't want to copy our node JS application into the image. We want to run NPM Install to build our application. Our application runs on port 5,000 so we would want to export that port. And once everything is built up, we want to launch up using the command node index.js. So this app, when you do an NPM Install, creates a file call index.js and that is the one which needs to be launched up to run the application, and that's what we are doing in here. The specifics of what we are doing in here are not really important. At high level, the important thing for you to understand is that in a Docker file, there are a sequence of steps where we define step by step how to create a Docker image. This is the base image that's basically the starting point into which we are copying a directory, we are building something, and once everything is built up, and at the end, we are saying this is the command which needs to be used to launch up a container when we create a container using this Docker image. Let's do a quick review of the instructions that we see in here. From is used to set up base image. Working directory sets the working directory. So what is the working directory to use for this subsequent set of commands? Run executes a specific command. Over here we are running NPM Install. Expose informs Docker about the port that the container listens on at run time. This node application that we are using will be running on port 5,000 and that's what we are telling Docker. We are saying this application would be running on port 5,000. Copy over here is used to copy files. What we are doing in here is we are copying files from our local machine into the image. So copy is used to copy new files or directories into an image, and CMD is the default command for executing a container. When this image is used to create a container, this is the command which would run to start up the application which is running as part of your container. What we are looking at is a very simple Docker file. There are a few best practices that you need to understand when it comes to building Docker images. You need to keep your images lightweight. The size of your image should be as small as possible. If you look at the Ubuntu Docker image, you'd see that it's huge. And that's why there are a lot of images with the name as Alpine which are created which are lightweight. That's the reason why you need to prefer Alpine images to other images. You can see that over here as well. We are making use of an Alpine image. The next important thing is to keep the images lightweight. Do not copy anything that's unnecessary into the image. So if you have things like node modules, so if you have worked with node JS then you know that node modules is not really needed to run the application. You need it to build the application but it is not needed to run the application. Similar to that, if you have worked with Java applications, there are a lot of files which are created in the target directory. You will need to run the JAR, but other than that, rest of the things which are created in the target directory you might not need them at all. So only copy the important things that you would need into the Docker image. Don't copy everything. One of the best practice when you are creating the instructions to create Docker image, that's basically your Docker file, is to ensure that the things which are present at the top change very less frequently. Why is it important? The easiest way to understand is to think as if for each of these commands after it is executed, a differently layer is created. So after executing this command, a layer is created. After executing this command, another layer is created. And so on and so forth for the rest of the commands. And as long as things don't change in that specific layer, Docker will not rebuild that layer, but it'll reuse the existing layer. So Docker checks when it's building this image again, has this changed? No, I'll reuse the image. Has this changed? Nope. I'll reuse the image. So as long as things don't change, Docker would reuse the layers up 'till then, and then it'll build the rest of

the steps. To reduce the time taken to build a Docker image, what you can do is to ensure that your Docker file promotes reuse of as many layers as possible. So the more layers that you can reuse, the lesser time it would take to build your Docker image. One important thing to understand is what changes and what does not change. If you look at your projects, typically, your code changes very often. For example, in this set of files, the code is presented index.js. Index.js changes very often, but dependencies change very less. Package.json is where the dependencies are defined. What are the libraries that we would need to run this node application? All these are defined in package.json. Package.json does not change as often as index.js. If you're familiar with Java, package.json is very similar to POM.XML. If you're familiar with Python, then package.json is very similar to your requirements.txt file where you define what libraries you want to make use of. These files, package.json or POM.xml, or requirements.txt do not change as often as your code. And that's the reason why one of the best practices is to move the copy of this file and the building of dependencies into a separate layer. And once that layer is built, you can copy your code. That would ensure that most of the scenarios where package.json does not change, you can reuse most of the layers up 'till then, and only the code layer, that's the last one, is the one which would change, and you'd be able to build your Docker images very quickly. And that's what we are seeing in here. Instead of copying the entire current folder, what we are doing is we are copying just the package.json into the app folder first. Instead of copying all the files, what we are doing is we are just copying package.json alone. So package.json is copied and we are running NPM Install. And after that is when we are copying the rest of the files into the app folder. So after that is when the index.js, that's the code, and maybe you will have source folders in here. That's when all these are copied into the app folder. So if you are making changes only to your index.js, only to your code or your source folders, all these layers up 'till here will be reused. Because we are not making changes to package.json, we don't really need to do NPM Install again so this layer also will be reused and only these layers would be rebuilt again and again. In this step, we got a 10,000 feet overview of creating Docker images and the best practices in creating Docker images. As far as the exam goes, you don't really need to know every detail related to creating a Docker image. You need to understand high level that a Docker file contains a set of instructions and these instructions are what are used to build a Docker image. And you also need to understand that you need to keep the Docker image size as small as possible. And that's the reason why we prefer using lightweight images like Alpine and we don't really need to copy anything that's unnecessary into your Docker image, and you need to ensure that when creating Docker images, you follow a proper layering. Things that do not change often should be at the top so that that layer is reused. Things we change often, you can keep it as a bottom layer.

Welcome back. In this step, let's look at some of the important scenarios related to Google Kubernetes Engine (GKE). Let's get started with the first one. You want to keep your cost low and optimize your GKE implementation. What are the options that you can think of? Some of the options that you can use are Preemptible VMs. Make sure that you're choosing the right region and make sure that you opt for committed. Use discounts if your workloads are, if you have continuous workloads that run for long period. Also, remember that E2 machine types are cheaper than N1 machine types for most workloads. So try and play with E2 machine types and see if they work out to be cheaper for you. Because Google Kubernetes Engines make use of Compute Engine virtual machines, the first thing that we are talking about in here is to optimize your usage of virtual machines. The next important thing is also to make sure that you choose the right environment to fit your workload type. If the workload type you are running needs GPUs, you need to ensure that your virtual machines that are part of your node pools have GPUs. One of the flexibilities that Kubernetes provides is you can have multiple node pools, so you can have a node pool of normal machines and you can have another node pool with GPU attached machines. You can deploy the workloads or the deployments that need the GPUs to the node pool with which GPUs are attached. You want an efficient, completely autoscaling GKE solution. Inside a Google Kubernetes cluster, you have a number of node pools. The node pools contain nodes. Nodes are where you have instances of your applications or deports that are deployed. And if you want complete autoscaling, you need to use HorizontalPodAutoscaler for deployments so you want to be able to increase the number of instances of your deployments automatically based on the load and that is done by using HorizontalPodAutoscaler. The command, which we saw earlier, was Kubectl Deployments Autoscale, it is not sufficient if you autoscale the instances of your deployment. The deployment should have sufficient nodes and how do you autoscale node pools? The way you can do that is by using a Cluster Autoscaler. You want to execute untrusted third party code in a Kubernetes cluster. Let's say you are doing some trial and you have some untrusted third party code and you want to run it as part of your Kubernetes cluster, one of the most important things to remember whenever you are running anything in Kubernetes cluster is if you have untrusted code. The best option is to actually create a separate node pool and run it there, so if you have any different kind of workload create a node pool and run it there so that it does not impact the other workload, which are running as part of your Kubernetes cluster. And over here as well, the best option would be to create a new node pool with something called a GKE Sandbox. GKE Sandbox allows you to run the untrusted code inside a sandbox. Once you have the new node pool, you can actually deploy the untrusted code to the Sandbox node pool. You want to enable only internal

communication between your microservice deployments in a Kubernetes cluster. If you don't want to expose your deployment to the outside world, you need a service. So the question is, what kind of service would you go for? Because I would want only internal communication over here, the service type I would choose is cluster IP. Whenever you choose cluster IP, only the services which are part of the Kubernetes cluster can talk to this service. Anything outside the cluster, will not be able to talk to this specific service. The next one is "my pod stays pending" so I'm trying to create a deployment and I see that my pod status remains pending. Most probably, the pod cannot be scheduled onto a node. If I say I need a hundred instances of a pod and let's say there are not sufficient resources available, there are not sufficient number of nodes available, then the pod cannot be scheduled onto a node and that's why your pod might stay pending. So what you do in this situation, you would increase the number of nodes in your node pool. "My pod stays waiting", this is a different status. Earlier it was pending, now it is waiting. What could be the possible reason? Most probable failure is because there is a problem with pulling the image. Whenever we create a deployment, we specify the image, the container image to make use of and if the path of the container image is not correct then you'll not be able to pull the image. The other problem might also be related to access. I don't have enough access to pull the image from the container repository. In both those situations, your pod would stay waiting. In this step, we looked at some of the most important scenarios related to Google Kubernetes Engine. I'm sure you're having a wonderful time and I'll see you in the next step.

Welcome back. Let's now delete all the resources that we have created in this specific section. To make it really, really interesting, what we'll do is we'll actually delete everything step by step. So, let's start with deleting the service. So kubectl, delete service, and the name of the service that we have created is Hello World Rest API. So, let's start with deleting the service first. That's cool. So, this would delete the service and the load balancer that is associated with that specific service. Once we delete the service, what we need to deploy delete, after the service, we need to delete the deployment. And after the deployment, what you can do is we can delete the cluster and after that what we can do is delete the project. So, to delete the service and the deployment, we'll be using kubectl. To delete the cluster, what we'll be using, Gcloud. So kubectl, delete deployment, Hello World Rest API. So, let's do that. So kubectl, delete deployment. Deployment is also now deleted. The next thing I would want to do is delete the cluster Gcloud container, clusters, delete. I would need my cluster and the zone in which I created the cluster was half and half in zone. US one, US Central 1C, oops I've not said the project yet. So Gcloud, I'll do a Gcloud projects list to list the different projects and I'll pick up the project ID and let's set it down here. So Gcloud project, Gcloud config, set project and this. And now I can go ahead and delete my cluster. So, it says the following clusters would be deleted my cluster in US Central one C. Yes, I would want to do that and this would go ahead and delete my cluster. The deletion of the cluster would take a long time. I would not really wait for it. So, what I would recommend you to do is to actually wait for the deletion of the cluster, and if you'd want you can even shut down the project under which we created the cluster. In this, we deleted all the things that we have created while learning about Kubernetes. I'm sure this was an interesting journey about Kubernetes, and I look forward to seeing you in the next section. Until then, bye bye.

Welcome back. In the last few steps, we learnt a lot of Kubernetes terminology, in a very hands-on way. In this step, let's review some of the important Kubernetes terminology. Whenever we talk about Kubernetes, there are two parts. One is the hardware, the cluster, the master node, the worker node, node pools. Two, it's the software, the applications which are running, pods, deployment, services and things like that. Let's start with the hardware. Whenever you'd want to run anything in Kubernetes, you need to create a cluster, and the cluster contains a master node. The master node is the one which manages the cluster. If you'd want to deploy something to Kubernetes, you'd be talking to the master node. If you'd want to change something on the worker node, that's the duty of the master node as well. So the master node is completely responsible for everything in the cluster, and the worker nodes are where your workloads run, your applications, your pods, they run on the worker nodes. You also have a concept called node pool, where you can create a group of nodes in a cluster with the same configuration. What does that mean? Let's say I have a machine learning workload that I would want to run, and I would need GPUs for it. What I can do, is I can create a set of nodes with GPUs and create them as a node pool. And that node pool can be used to run your machine learning algorithms. All the rest of the things can be running in another node pool, which would be using the usual configuration. So node pools helps you to group worker nodes. You can create worker nodes with different kinds of configuration. So that's the hardware part, master nodes, worker nodes and node pools. Let's now move on to the software part. Can you directly create a container and deploy it into Kubernetes? Nope. Containers always run as part of something called a pod. Pod is the smallest deployable unit in Kubernetes, and pods are deployed. Pods are running on the worker nodes. Can you directly create a pod? The answer is no. You need to create a deployment. Deployment is the thing which manages the pod. You would not need one pod, for the same application, I might have 10 pods, 20 pods, or 30 pods. Deployment is the one which manages that. Deployment will be able to auto-scale it as well, increase the number of instances, decrease the number of instances, and it also provides you a number of features around release management, checking whether a pod is up and running, and a lot

of other things. Deployments are used to manage pods. And the last concept that we'll be touching upon in this specific step is service. You might have applications running in your pods, you'd want to expose them out to the outside world. That's where you would go for a service. The idea behind this step is to quickly review some of the important Kubernetes terminology that we have learnt in the last few steps. I'm sure you're having a wonderful time, and I'll see you in the next step.

10 - Getting Started with Google Cloud Functions
-: In this step, let's get started with Cloud Functions. Imagine you want to execute some code when an event happens, when a file is uploaded to the object storage service in Google Cloud, which is cloud storage. Or an error log is written to the logging service in Google Cloud, which is cloud logging. Or a message arrives in a queue, for example Google Cloud Pub/Sub. Or somebody invoked a HTTP or a HTTPS URL. So an event happened and you want to execute some code in response. That's where you can make use of Cloud Functions, run code in response to events. You can write your business logic in Cloud Functions in multiple languages, Node.js, Python, Go, java.net, Ruby, and a number of other languages are supported. When you're using Cloud Functions, you don't need to worry about servers scaling or availability. All that you need to worry about is your code. Even if you get millions of messages on the queue, your Cloud Functions would automatically scale to process the messages which are coming in. So you don't really need to worry about scaling or availability. Or you don't need to worry about the fact that there are servers in the background. That's what Cloud Functions is all about. When you're using Cloud Functions you pay only for what you use. You pay for the number of invocations which are coming into your Cloud Function. You'd pay for the compute time of your invocations. How long did the invocation run for? And you'll pay for the amount of memory and CPU that you have provisioned for your Cloud Function instance. Cloud Functions are time bound. The default timeout is one minute or 60 seconds and you can configure a timeout of up to 60 minutes. That is a max of 3,600 seconds. There are two product versions of Cloud Functions. You have the first generation Cloud Functions which is the first version of Cloud Functions and right now we also have the second generation of Cloud Functions. This is the new version of Cloud Functions which is built on top of Cloud Run and Eventarc. In this step, we got introduced to Cloud Functions. I'm really excited to help you learn a lot more about Cloud Functions. I'll see you in the next step.
Welcome back. In this step, let's look at some of the important concepts related to Cloud Functions. An event is something that happened. So you uploaded a object to Cloud storage. An HTTP call is made. Or a message arrived in Pub/Sub. So these are all events and trigger is the response to an event with a function call. So which function to trigger when an event happens. That's the trigger. What do the functions do? The functions take the event data and perform the action that you'd want to perform. Events for Cloud Functions can be triggered from a variety of places. Object storage service in Google Cloud, which is called Cloud Storage. Cloud Pub/Sub, which is a popular queuing service in Google Cloud Platform. Or, you can also trigger it from HTTP calls. A HTTP call arrives and you can call a Cloud Function in response to a HTTP call. You can also trigger your events from Firebase, Cloud Firestore, and stack level locking. The important concept to understand are events, triggers, and functions. Event is something happening. Trigger is making sure that you respond to an event with a function call. And function is what gets the event data and performs the action. The functions can be triggered on events from a variety of places. I'm sure you're having an interesting time and I'll see you in the next step.
Welcome back. Let's log into Google Cloud console and try and create our first cloud function. I am in my first project, so make sure that you have switched over to my first project. If you have not deleted or shut down your Kubernetes project, make sure that you are switching over to your my first project, and you can actually search for cloud functions. So cloud functions is what we are looking for. The cloud functions are getting enabled. So the first thing is the API should be enabled. So that's what is happening right now. And as it says in here, Google Cloud functions is a lightweight, event-based, asynchronous compute solution that allows you to create small single purpose functions that respond to cloud events without the need to manage a server or a runtime environment. Okay, a long definition. But I guess most of it made sense. Let's go ahead and say create function. Okay, let's wait for a function to be created. Now the first choice that you'd need to make when you are creating a cloud function is whether you'd want to go for first gen or second gen. As you can see right now second gen is in preview, but I expect it to be live by the time you're watching this. Now, what is the difference between first gen and second gen? Second gen provides you with a lot of advanced features. If you choose first gen, and if you scroll down and choose runtime build connection and security settings, and go further down. If you look at the timeout which is default 60 seconds. But if you hover over there, you can see that the maximum timeout for a cloud function of version V1 is 540 seconds. So the maximum amount of duration for a specific invocation of a specific cloud function is only nine minutes. And the maximum amount of memory that you can allocate is 8 GB. However, you'd see that it would be a little different when you go for a second gen. So if I go for second gen function, you can see that you can configure up two 16 GB. And more importantly, if your cloud function is triggered through HTTPS, then the maximum timeout can be up to 60 minutes. You can see that this is a significant increase. From nine minutes, we are going to 60 minutes. For now, let's keep things simple as second gen is still in preview. I would want you to get a consistent experience. So let's go with first gen. What we'd do is configure the function name. I'll call this my first cloud function. I'll choose the default region which is suggested. You can copy the URL, so that we can send request to it a little later. Over here, you can also see that you can configure the trigger type. Whether you'd want to trigger it through HTTP or you'd want to trigger it from Cloud Pub/Sub, or from cloud storage. Where do you want to trigger

it from. I'll go for HTTP. And you can also configure how you'd want to authenticate requests. For now, let's choose allow unauthenticated invocations. I don't want to worry about HTTPS for now, so I would uncheck this box. And I would say Save. I'll minimize the runtime build connections and security settings, and I'll click Next. On next screen, you can configure this specific code that you'd want to use to run your cloud function. You can configure the runtime you would want to make use of. As you can see in here, .NET is supported. Go, Java, Node, Python, PHP, Ruby. A lot of languages are supported in here. Let's choose the default one which is suggested, which is Node.js 16. And as you can see in here, it says Cloud build API is required to use the runtime selected. So I would go ahead and say Enable API. I'll go ahead and click enable in here. Cloud Build is a CI/CD tool, continuous integration continuous deployment tool, and it is used to deploy a cloud function. And you need Cloud Build to be able to deploy a cloud function. So you can see that the Cloud Build API is now enabled. So I'll close this window, go back to the old window. And you'd see that there is no warning now. So over here in the inline editor, you can see a sample cloud function present. As you can see in here, it takes some of the information from the request and creates a response based on that. We don't really need to worry about what's in here. All that we need to say now is deploy. So this would create our first cloud function. The most important thing about cloud functions is the fact that you only pay for it when there are invocations to it. So the deployment of our cloud function is in progress. Okay, it took up about a couple of minutes, and the deployment was complete. And if I go into that specific function now, I can see the details about this specific function. Over here if I go over to testing, this is where you can actually test the function. So you can click test the function. And you'd see the response back from the function in here. If you'd want to actually add in logs, what we can do is go back to source. And I would say Edit. Open up this trigger URL in another window. So this is actually... Oops, let's click this. So this is another way you can actually invoke the function. So you can use this URL. And you can see that hello world is a response that is coming back. Now I would want to a change in the code. So I'll say Next. And over here, what I would want to do is print to the log. So console.log, and let's print the message. And let's say Deploy. So this would update the function and deploy the new version of the function. And you'd see that the URL that we used earlier would continue to work even while the function is getting deployed. Okay, it took about a minute, and the deployment is not complete. So I'll make a few invocations from here. And let's go back to the function. Right now, we have just created the function. So some of these data is not yet populated invocations per second. However, you can see some data about execution time. You can see that 95% of the requests were executed within 25 seconds. 99 percentile is 25.257 milliseconds. You can see that 99% of the requests executed within 25.257 milliseconds. So you can get a little bit of information about the execution time. You can see how much memories made used of. So 99% of the requests used less than 15.506. 50% of the requests made less than 15.144 MB. And you can see which zone occurred. You can see the number of active instances right now. You can see that there is just one instance which is active. And if there are any errors or things like that, you can look at them in here. If you want to look at the logs, you can go in here and look at the logs as well. So we triggered a function a few times. So you can see our hello world message printed in here. So this message is coming from Cloud Logging. Your cloud functions automatically integrates with Cloud Logging. And you can see the logs in here. Similar thing will happen even if you go to testing and test it now. If you go to testing and you test the function, what would happen? You'd see output. And if you wait for a little while and scroll down, you should see also the logs related to that specific invocation. It took a minute or so. And then I can see the logs related to that specific invocation in here. In this step, we played around with cloud functions.