[ENGLISH]

Hello,

Cloud Gurus and welcome to this lecture where we're going to cover the various

different options for updating applications running in Elastic

Beanstalk.

Now there are 4.

Different options and each one has its own advantages and trade-offs

which you will need to be aware of.

So the first option is an all at once deployment,

which is the simplest.

Approach, because it deploys.

To all hosts concurrently.

Next, we have rolling updates, which deploys.

The new version in batches instead of deploying to everything.

All at once. There's also rolling with additional batch,

which.

Launches an additional batch of instances,

and then deploys the new version of your application in batches.

We also have immutable deployments, which.

Deploys the new version of your application to a fresh group of instances

before deleting the old instances.

And then finally we have 1 more deployment policy and it was announced a good

few years after all of the others.

And this is the most recent one that they announced,

and it may or may not come up in the exam. It's just good to be aware of,

and this one is called Traffic Splitting and this installs a new

version of your application on a new set of instances,

just like an immutable deployment,

but then it forwards just a percentage of the incoming client traffic

to the new application version for evaluation.

So you've got some traffic going to your old version and some traffic going to

your new version.

And this allows you to test the new version in production before fully

committing. So which approach should you choose and why?

Well it all depends on the needs of your application.

So let's dig.

Deeper into each of these strategies and hopefully it will all become clear.

Beginning with the.

All at Once method.

So here is our group of instances in our Elastic Beanstalk environment,

and they are running version 1 of our application.

And with an All at.

Once deployment, it's going to deploy to all instances simultaneously.

So you will experience a total outage while the deployment takes

place.

Therefore.

This is not ideal for mission critical production systems

or applications, which needs to be available 24/7. So think of amazon.com,

for example.

They're not going to be able to use this deployment approach because the website

is going to be offline for the duration of the deployment.

And then if the update fails.

Say, there's an issue with version 2 of your application.

You've got a big problem.

You're going to need to roll back the changes by redeploying the original

version to all of your instances,

which is going to result in another outage to get you back to the previous

version. So when would you use the all at once deployment approach?

Well,

I think the only time you would use it is in a development or test environment,

just because it's a really quick and simple way to do things.

But if I was looking after a production environment,

this is not the kind of approach that I would want.

To take. So what about a rolling deployment? Well, this is where.

Elastic Beanstalk deploys the new version in batches.

Then each batch is taken out of service while the deployment takes place.

So here's our 2 batches. We upgrade the first batch.

Then we upgrade the second batch. And with this approach,

you won't experience an outage. However,

your environment capacity will be reduced by the number of instances in a

batch while the deployment is taking.

Place. So this is.

Not going to be ideal for performance sensitive systems, and in this example,

we've reduced our capacity by 50% while we were deploying the new

version. However,

it could be a good option if you want to keep your service up and running during

the deployment, but performance is not an important factor.

And if you think about a website for your local library, for example,

they're not going to have massive performance requirements.

So this might be a good option for that kind of website.

But if.

You're running a busy e-commerce website,

then it's probably not going to be a great option.

And what about if our rolling deployment fails? Well, if it fails,

you're going to need to perform an additional rolling update to roll back the

changes.

Moving on to Rolling with Additional Batch and with this approach,

Elastic Beanstalk launches an additional batch of instances.

So here's our new batch of instances.

Then it deploys the new version in batches,

and this enables you to maintain full capacity throughout the deployment.

Now, if an update fails, once again.

You'll need to perform an additional rolling update to roll back the changes.

Now, I actually.

Think that rolling with additional batch is quite a good option because you

won't experience any degraded performance during the deployment process.

However, rolling back after a failure is still going to take some time.

And if your application is highly critical, it needs to be running 24/7,

and you cannot tolerate waiting to roll back changes if there's a problem with

your version 2 code, then this is not going to be a suitable approach.

So this is where immutable deployments come in and the word immutable

means unchangeable, so something that cannot be changed or modified.

And with an immutable deployment,

this takes the approach that our existing deployment must not be changed.

Therefore, we deploy the new version to a fresh group of instances,

and there is our fresh group of instances with version 2 of our application,

and only when the new instances pass their health checks should the old

instances be terminated.

And for me.

This is my preferred approach because it doesn't impact the live environment

in any way at all.

You can run your deployment independently from your live application,

and if something goes wrong with a new version of the application you just

deployed,

then all you need to do is delete the new instances and make sure your load

balancer is pointing to the original instances.

So rolling back is going to be pretty much immediate and using an

immutable deployment is the preferred approach for mission critical systems.

And then finally we have traffic splitting and I see this really as an

extension of an immutable deployment and traffic

splitting allows you to perform Canary testing during your

application deployments.

And if you haven't heard about the Canary in the coal mine,

it's an old practice where coal miners would bring a poor Canary

down in to the mine, tunnels with them while they were working,

so if any dangerous gases like carbon monoxide built

up within the mine, then the Canary would.

React. So maybe it would pass.

Out or start making noises,

and this would alert the miners so that they could evacuate

immediately. And if you're an animal lover like me,

then don't be alarmed because they would also take a tank of oxygen to

revive the little Canary if it passed.

Out. And traffic.

Splitting absolutely enables, what's known as Canary testing.

So you can test out your new deployment in production by directing a

small percentage of traffic towards your new application version.

So how does it work? Well.

First of all,

Elastic Beanstalk will install the new version of the application on a

new set of instances, just like an immutable deployments,

but then it builds on this idea of an immutable deployment by forwarding

a percentage of incoming client traffic to the new application version

for a specified evaluation period. So for example,

you might be sending 90% of your traffic to version 1 of the

application, your stable version,

and then send the remaining 10% to version 2,

the new version of your application.

And then if the new instances stay healthy,

then Elastic Beanstalk will forward 100% of the traffic to the new

instances and terminate the old ones.

And just like the Canary in the coal mine,

this provides an early indication that something is.

Wrong. So for.

The exam,

you'll need to understand the differences between each type of deployment.

They'll give you a scenario and you will need to identify the best.

Option. So we've got the All at Once approach,

which deploys.

The new version of your application to all your instances simultaneously

it involves a service interruption and rolling back is going to require a

further All at Once update and for my money, this is the worst option.

It's simple and it's okay for development and test, but for production,

I would definitely avoid it.

Then we've got the rolling update where we're.

Upgrading in batches. And with this option,

you've got a reduced capacity during the deployment and rolling back requires a

further rolling update.

So this 1 is only going to be suitable for applications which are not

performance sensitive. So think about the website for your local library,

which is probably not going to be performance.

Sensitive. We've.

Then got rolling with additional batch, which I don't think this is too bad,

an option because it allows you to maintain full capacity during the deployment.

However, rolling back is going to require a further rolling update.

So it's not the best option if you need to be able to roll back very.

Quickly.

Which brings us to the immutable deployment.

And this is where we're deploying on a new set of instances and we don't touch

our existing live environment at all.

So this is my favorite one because it allows you to maintain full capacity.

We're not touching our live environment.

In any way. And to roll back.

All we do is delete the new instances we created and make sure our load

balancer is pointing to our original application version.

And this one is the preferred option for mission critical production systems,

because you've got no downtime. You've got no reduction in capacity.

And rolling back to a previous version is pretty much instantaneous.

And then.

Finally the traffic splitting deployment option,

which performs an immutable deployment and then splits the traffic

between the old and the new deployment, allowing you to perform Canary.

Testing. And this is.

Of course,

another preferred option for mission critical production systems,

but it's a relatively new deployment option and it may or may not come up in the

exam. So if you're running mission critical systems,

then immutable or traffic splitting are the best options

and All at Once is probably the worst.

Option. So that.

Is the end of this lecture. If you have any questions, please let me know.

Otherwise, feel free to move on to the next lecture. Thank you.

[SPANISH]

Hola,

Cloud Gurus y bienvenidos a esta conferencia en la que cubriremos los diversos

diferentes opciones para actualizar aplicaciones que se ejecutan en Elastic

habichuelas mágicas.

Ahora hay 4.

Diferentes opciones y cada una tiene sus propias ventajas y desventajas

lo que tendrá que ser consciente.

Entonces, la primera opción es una implementación completa,

cual es la mas sencilla.

Acercamiento, porque se despliega.

A todos los hosts al mismo tiempo.

A continuación, tenemos actualizaciones continuas, que se implementan.

La nueva versión en lotes en lugar de implementar en todo.

De repente. También está rodando con lote adicional,

cual \_

Lanza un lote adicional de instancias,

y luego implementa la nueva versión de su aplicación en lotes.

También tenemos implementaciones inmutables, que.

Implementa la nueva versión de su aplicación en un nuevo grupo de instancias

antes de eliminar las instancias antiguas.

Y finalmente tenemos 1 política de implementación más y se anunció una buena

pocos años después de todos los demás.

Y este es el más reciente que anunciaron,

y puede o no aparecer en el examen. Es bueno ser consciente de,

y este se llama Traffic Splitting y esto instala un nuevo

versión de su aplicación en un nuevo conjunto de instancias,

como un despliegue inmutable,

pero luego reenvía solo un porcentaje del tráfico entrante del cliente

a la nueva versión de la aplicación para su evaluación.

Así que tienes algo de tráfico yendo a tu versión anterior y algo de tráfico yendo a

tu nueva versión.

Y esto le permite probar la nueva versión en producción antes de

cometiendo \_ Entonces, ¿qué enfoque debería elegir y por qué?

Bueno, todo depende de las necesidades de su aplicación.

Así que vamos a cavar.

Profundice en cada una de estas estrategias y, con suerte, todo quedará claro.

Comenzando con el.

Método todo a la vez.

Este es nuestro grupo de instancias en nuestro entorno de Elastic Beanstalk,

y están ejecutando la versión 1 de nuestra aplicación.

Y con un Todo en.

Una vez implementada, se implementará en todas las instancias simultáneamente.

Por lo tanto, experimentará una interrupción total mientras se lleva a cabo la implementación.

lugar \_

Por lo tanto.

Esto no es ideal para sistemas de producción de misión crítica.

o aplicaciones, que debe estar disponible 24/7. Así que piensa en amazon.com,

por ejemplo

No podrán usar este enfoque de implementación porque el sitio web

estará fuera de línea durante la implementación.

Y luego si la actualización falla.

Digamos que hay un problema con la versión 2 de su aplicación.

Tienes un gran problema.

Tendrá que revertir los cambios volviendo a implementar el original

versión a todas sus instancias,

lo que resultará en otra interrupción para volver al anterior

versión \_ Entonces, ¿cuándo usaría el enfoque de implementación todo a la vez?

Bien,

Creo que la única vez que lo usaría es en un entorno de desarrollo o prueba,

simplemente porque es una forma muy rápida y sencilla de hacer las cosas.

Pero si estuviera cuidando un entorno de producción,

este no es el tipo de enfoque que me gustaría.

Tomar. Entonces, ¿qué pasa con una implementación continua? Bueno, aquí es donde.

Elastic Beanstalk implementa la nueva versión en lotes.

Luego, cada lote se pone fuera de servicio mientras se lleva a cabo la implementación.

Así que aquí están nuestros 2 lotes. Actualizamos el primer lote.

Luego actualizamos el segundo lote. Y con este enfoque,

experimentará un apagón. Sin embargo,

de su entorno se verá reducida por el número de instancias en un

lote mientras se lleva a cabo la implementación.

Lugar. Entonces esto es.

No va a ser ideal para sistemas sensibles al rendimiento y, en este ejemplo,

hemos reducido nuestra capacidad en un 50 % mientras implementábamos el nuevo

versión \_ Sin embargo,

podría ser una buena opción si desea mantener su servicio en funcionamiento durante

el despliegue, pero el rendimiento no es un factor importante.

Y si piensa en un sitio web para su biblioteca local, por ejemplo,

van a tener requisitos de rendimiento masivos.

Así que esta podría ser una buena opción para ese tipo de sitio web.

Pero si.

Está ejecutando un sitio web de comercio electrónico ocupado,

entonces probablemente no será una gran opción.

¿Y qué pasa si nuestra implementación continua falla? Bueno, si falla,

necesitará realizar una actualización continua adicional para revertir la

cambios \_

Pasando a Rolling with Additional Batch y con este enfoque,

Elastic Beanstalk lanza un lote adicional de instancias.

Así que aquí está nuestro nuevo lote de instancias.

Luego implementa la nueva versión en lotes,

y esto le permite mantener la capacidad total durante la implementación.

Ahora, si una actualización falla, una vez más.

Deberá realizar una actualización continua adicional para revertir los cambios.

Ahora, en realidad.

Piensa que rodar con lote adicional es una muy buena opción porque

no experimentará ningún rendimiento degradado durante el proceso de implementación.

Sin embargo, revertir después de una falla aún llevará algún tiempo.

Y si su aplicación es muy crítica, debe estar funcionando las 24 horas del día, los 7 días de la semana,

y no puede tolerar esperar para deshacer los cambios si hay un problema con

su código de la versión 2, entonces este no será un enfoque adecuado.

Aquí es donde entran las implementaciones inmutables y la palabra inmutable

significa inmutable, por lo que algo que no se puede cambiar o modificar.

Y con un despliegue inmutable,

esto toma el enfoque de que nuestra implementación existente no debe cambiarse.

Por lo tanto, implementamos la nueva versión en un nuevo grupo de instancias,

y ahí está nuestro nuevo grupo de instancias con la versión 2 de nuestra aplicación,

y solo cuando las nuevas instancias pasen sus controles de salud, las antiguas

las instancias sean terminadas.

Y para mí.

Este es mi enfoque preferido porque no afecta el entorno en vivo.

de ninguna manera en absoluto.

Puede ejecutar su implementación independientemente de su aplicación en vivo,

y si algo sale mal con una nueva versión de la aplicación, simplemente

desplegado ,

entonces todo lo que necesita hacer es eliminar las nuevas instancias y asegurarse de que su carga

balancer está apuntando a las instancias originales.

Entonces, retroceder va a ser bastante inmediato y usar un

inmutable es el enfoque preferido para los sistemas de misión crítica.

Y finalmente tenemos la división del tráfico y lo veo realmente como una

extensión de un despliegue inmutable y tráfico

la división le permite realizar pruebas Canary durante su

de aplicaciones .

Y si no has oído hablar del canario en la mina de carbón,

es una vieja práctica donde los mineros del carbón traerían un canario pobre

hacia la mina, túneles con ellos mientras trabajaban,

así que si se acumulan gases peligrosos como el monóxido de carbono

dentro de la mina, entonces el Canario lo haría.

Reaccionar. Así que tal vez pasaría.

Salir o empezar a hacer ruidos,

y esto alertaría a los mineros para que pudieran evacuar

inmediatamente \_ Y si eres un amante de los animales como yo,

entonces no te alarmes porque también llevarían un tanque de oxígeno para

revive al canario si paso.

Afuera. y trafico

La división permite absolutamente lo que se conoce como prueba Canary.

Para que pueda probar su nueva implementación en producción dirigiendo un

pequeño porcentaje de tráfico hacia su nueva versión de la aplicación.

¿Entonces, cómo funciona? Bien.

En primer lugar,

Elastic Beanstalk instalará la nueva versión de la aplicación en un

nuevo conjunto de instancias, al igual que una implementación inmutable,

pero luego se basa en esta idea de una implementación inmutable al reenviar

un porcentaje del tráfico de clientes entrantes a la nueva versión de la aplicación

durante un período de evaluación específico. Así por ejemplo,

podría estar enviando el 90% de su tráfico a la versión 1 del

aplicación , su versión estable,

y luego enviar el 10% restante a la versión 2,

la nueva versión de su aplicación.

Y luego, si las nuevas instancias se mantienen saludables,

luego, Elastic Beanstalk reenviará el 100 % del tráfico al nuevo

instancias y terminar las antiguas.

Y como el canario en la mina de carbón,

esto proporciona una indicación temprana de que algo es.

Equivocado. Entonces para

El examen,

deberá comprender las diferencias entre cada tipo de implementación.

Le darán un escenario y tendrá que identificar el mejor.

Opción. Así que tenemos el enfoque Todos a la vez,

que se despliega.

La nueva versión de tu aplicación a todas tus instancias simultáneamente

implica una interrupción del servicio y la reversión requerirá una

Además, la actualización All at Once y, por mi dinero, esta es la peor opción.

Es simple y está bien para desarrollo y prueba, pero para producción,

Definitivamente lo evitaría.

Luego tenemos la actualización continua donde estamos.

Actualización en lotes. Y con esta opción,

tiene una capacidad reducida durante la implementación y la reversión requiere un

más actualización continua.

Entonces este 1 solo será adecuado para aplicaciones que no son

al rendimiento . Así que piense en el sitio web de su biblioteca local,

que probablemente no va a ser el rendimiento.

Sensible. tenemos

Luego se puso en marcha con un lote adicional, que no creo que sea tan malo,

una opción porque le permite mantener la capacidad total durante la implementación.

Sin embargo, la reversión requerirá una actualización continua adicional.

Por lo tanto, no es la mejor opción si necesita poder retroceder mucho.

Rápidamente.

Lo que nos lleva al despliegue inmutable.

Y aquí es donde estamos implementando un nuevo conjunto de instancias y no tocamos

nuestro entorno en vivo existente en absoluto.

Así que este es mi favorito porque te permite mantener la capacidad máxima.

No estamos tocando nuestro entorno en vivo.

De cualquier manera. Y para retroceder.

Todo lo que hacemos es eliminar las nuevas instancias que creamos y asegurarnos de que nuestra carga

balancer apunta a nuestra versión original de la aplicación.

Y esta es la opción preferida para los sistemas de producción de misión crítica,

porque no tienes tiempo de inactividad. No tienes reducción en la capacidad.

Y volver a una versión anterior es prácticamente instantáneo.

Y luego.

Finalmente, la opción de implementación de división de tráfico,

que realiza una implementación inmutable y luego divide el tráfico

entre el antiguo y el nuevo despliegue, permitiéndote realizar Canary.

Pruebas. Y esto es.

Por supuesto,

otra opción preferida para los sistemas de producción de misión crítica,

pero es una opción de implementación relativamente nueva y puede o no aparecer en el

examen \_ Entonces, si está ejecutando sistemas de misión crítica,

entonces inmutable o la división del tráfico son las mejores opciones

y All at Once es probablemente el peor.

Opción. De modo que.

Es el final de esta conferencia. Si tiene alguna pregunta, por favor hágamelo saber.

De lo contrario, siéntase libre de pasar a la siguiente lección. Gracias.