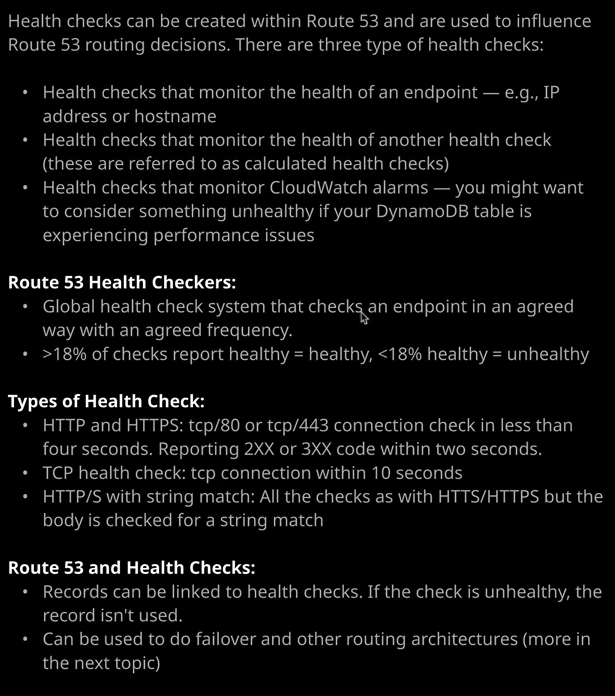
Route 53 Health Checks

Now**, Route 53 has health checkers in locations around the world and you can think of these as servers that are configured to perform health checks against endpoints that you provide to Route 53.** So these are geographically dispersed, they're positioned globally, and you essentially create a health check, you provide end point details, tell it whether you want that check to occur every 10 or every 30 seconds and those checkers perform health checks that you specify on these endpoints.



Now Route 53 is actually capable of doing a number of different type of health checks. If I go to health checks and a create a health check, it can do an **endpoint check**, and I'll talk about that in a minute but essentially, an endpoint check actually checks the physical health of that endpoint based on whether it's HTTPD, HTTPS, TCP. You can specify a lot of different options, but essentially it directly monitors the health of an endpoint. We've also got a health check which could monitor the health of multiple other health checks. This is known as a **calculated health check**. Now what you might want to use this for is if you've got lots of different services maybe different components of your system and you're creating an individual health check for each of these components, you might want to create a health check that kind of summarizes the health of all these individual components. So maybe you've got your frontend tier, your logic tier, and your database tier, and each of those has got a health check well you can create a calculated health check that will check the status of these individual checks and then only report as healthy if maybe all of the subjects report as healthy or a certain number of them, so you get that additional flexibility. It's kind of like a meta health check you—So you're able to create this check that basically summarizes the health of other health checks and generates this summarized report of a system health. So it's a really flexible feature.

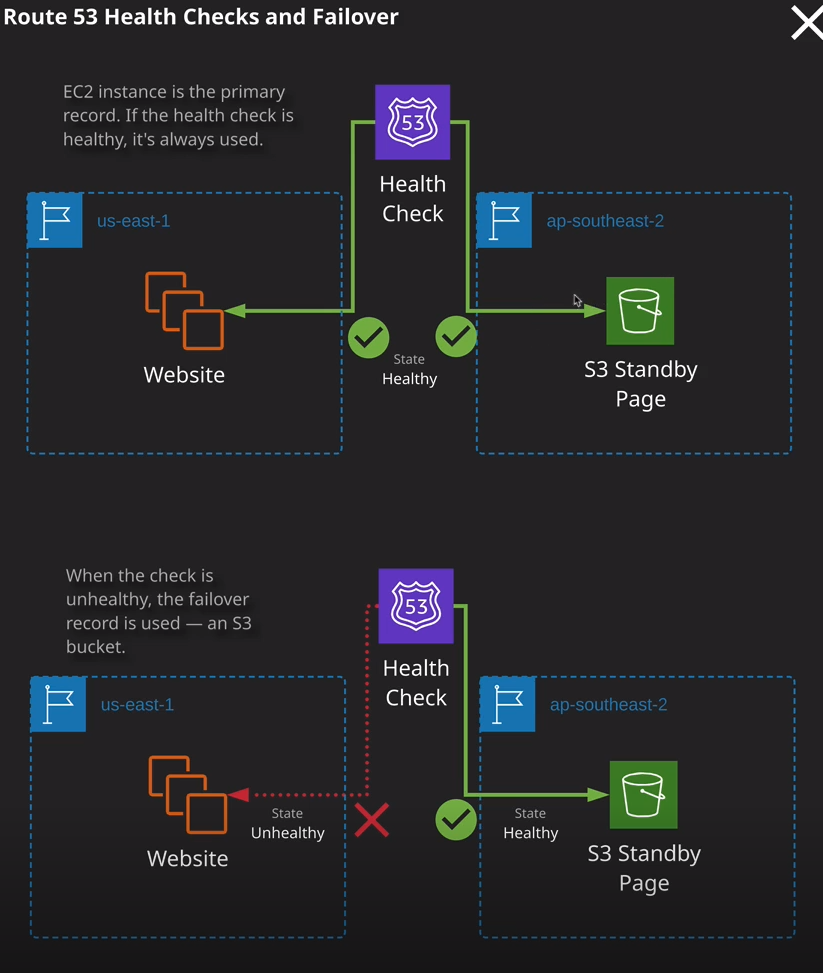
We've also got the third type, which is where you can check with state of a **CloudWatch alarm**. So maybe if a DynamoDB database is being throttled or other services are experiencing issues, you might want to use this type of health check if your system utilizes some underlying AWS components and you want to mark it as unhealthy if one of those components itself fails, what you don't know yet is where you use these health checks, and I'll be showing you that in the next topic of the course. **These health checks allow Route 53 to adjust its behavior to respond with different resources depending on the health check status and so, by being able to monitor CloudWatch alarms, you give yourself a lot more flexibility over which AWS resources you can involve in these checks**.

Now, the type of health check that's used most often is the simple endpoint check. **With an end point check, you can specify a resource either directly using an IP address or a domain name. If you specify a domain name it simply means that these health checkers will resolve the domain name to an IP address before they perform the health check and so, if you do have any IP addresses that regularly change but you have static domain names, then by doing this you could make sure that these health checkers I always get the right IP address.**

Now the checkers perform a number of different types of checks. We've got **HTTP or HTTPS checks** and essentially, this means that these health checkers are going to attempt to connect to an endpoint either on port 80 if you use HTTP or 443 if you use HTTPS and it's going to expect that endpoint to accept the connection within four seconds and report a 200 of 300 series status code within two seconds and if it does that it's going to treat it as healthy. Anything else and it will be viewed as unhealthy.

There's **also TCP and TCP can be used for applications,** which aren't HTTP or HTTPS and a TCP check expects to be able to connect to a specific port on a resource within 10 seconds and have that connection attempt accepted. Anything outside that it will be reported as unhealthy.

Now, the type of check the select influences the different options that you have available. So if you select a TCP check, you simply specify the IP address and the port. If you specify HTTP or HTTPS than you able to specify in this case, the IP address optionally a hostname and if you include this, then it's specified to the resource that's being checked. So this is useful If you've got different hostnames mapping to an individual IP address, maybe you're running three or four different websites that use host headers as part of the web server. So you might have Netflix.com and linuxacademy.com on the same server using the same IP address, and if you specify the host name in here, it will get passed through to that server. A**s well as the port you're also able to specify a path and that's specific to HTTP or HTTPS.** Now there's also some advance configuration. **The default frequency is that these checkers perform this check every 30 seconds. You can elect to specify a faster, more frequent check every 10 seconds. The failure threshold is the number of checks that need to fail before Route 53 deems this resource unhealthy when actually happens is all of the checkers check the resource and it's deemed to be healthy if more than 18% of the checks report as healthy, if less than 18% reports healthy, then is deemed to be unhealthy check and then after three of these checks, if three of them are unhealthy then the health check itself, reports as unhealthy, and this can obviously be set to different values.** Now keep in mind that if your request interval is 10 seconds and a failure threshold of three generally means an approximate 30 second response time to failure. So keep that in mind whatever you set this **failure threshold to be, it** will determine how quickly Route 53 can react to these failures. If it's a standard 30 seconds check then three checks is actually equal to 90 seconds. So be careful of how these two different settings interact. Now the default for HTTP is just to perform a basic check on the HTTP server with this path and if it gets a 200 or 300 status check, then it's viewed as healthy. You're also able to do **string matching**, which allows you to specify strings that are to be expected when this check completes. This allows you to do more advanced checking for failure or any error states in your application. **Now these things do come at extra cost**. We've got **latency graphs** which record the latency of all these status checks. So if you do want some information about how your application is responding over time, then you can use these latency graphs. There are some situations where you might want this to behave in a slightly different way. So at the moment, this check will do the check that you define and if it passes, it reports as healthy. If it fails, it reports as unhealthy. You can check this box to invert that check, so if this completes, then it responds is unhealthy, and that just gives you a little bit more flexibility. By default health checks come from a wide range of geographic sources. You are able to select customize and specify individual regions that you want the checks to come from but generally you should be **using recommended,** because that gives you a nice wide geographic spread and you won't be susceptible to any localized internet failures if you've got this widespread geographically, if you select customize and the limited to particular regions, you always face the possibility that you could be generating false alerts because of individual failures of these regions. So generally there's no great reason to adjust this from the default, stick to use recommended, and you'll get the best results.



Now, the architecture of health checks is pretty simple. You point at a particular resource, maybe a set of EC2 instances. If the check is healthy, it's a healthy state. If it's unhealthy, it reports as unhealthy and you can take other actions. Now one scenario that we'll look at in the next topic of the course is fail over routing and fail over routing allows us to have a primary resource, so maybe a website running on EC2 and if that ever reports is unhealthy, then it can direct the traffic towards a website that's hosted in S3. I won't be showing you that practically within this lesson, because I am just covering the theoretical concepts for now. In the next topic of the course, you'll be getting some good practical experience of how these health checks work because they're going to be used for each of the different routing types. Now health checks are pretty cheap. I generally don't like doing costing courses because things change so quickly, but at the time of creating this lesson, it costs 50 cents per health check per month for AWS resources and 75 cents for any non AWS resources. Additional features such as HTTPS or shrink matching or their checks every 10 seconds or recording any latency they all come extra cost, but it is a really good value feature that's available as part of Route 53. If you compare it to other commercial solutions, it is really aggressively priced. So unless you've got some requirements that you know you need a more fully featured monitoring application, then you should definitely give Route 53 health checks a look and they integrate really well with the different routing policies available in Route 53. Which is what we're going to look out in the next topic of the course. Now, with that being said, that is a great point to move on. We finished all of the lessons inside the DNS fundamentals topic of the course. In the next topic, which is advanced Route 53 we're going to look at the different routing policies available so simple, fail over, weighted, geographic, many more, and they generally use this health check functionality as well as some other advanced functionality that Route 53 provides, so I can't wait to get started. It's a really useful set of functionality to be aware of both of the exam and production usage. So go ahead, mark this video as complete and when you're ready, you can join me in the next section.